Composition automatique et adaptative de services web pour la météorologie

Benoît Gschwind

ED n° 84 : "Sciences et technologies de l'information et de la communication"

N° attribué par la bibliothèque

|__|__|__|__|__|__|__|__|__|__|__|__|__|

T H E S E

pour obtenir le grade de :
DOCTEUR DE L'ECOLE NATIONALE SUPERIEURE DES MINES DE PARIS
Spécialité "Informatique temps réel - Automatique - Robotique"

présentée et soutenue publiquement par
Benoît GSCHWIND

le 28 septembre 2009

Composition automatique et adaptative de services web pour la Météorologie

Directeurs de thèse : Robert MAHL et Lucien WALD

Jury :

Mme Valérie Roy (Maître de recherche ENSMP) ..... Présidente
Mme Anne Doucet (Professeur des Universités) ..... Rapporteur
Mr Dominique Dumortier (Professeur ENTPE) ..... Rapporteur
Mr Mohand-Said Hacid (Professeur des Universités) ..... Examinateur
Mr Robert Mahl (Professeur ENSMP) ..... Examinateur
Mr Lucien Wald (Maître de recherche ENSMP) ..... Examinateur

Remerciement

Cette thèse s'est déroulée à Mines Paristech conjointement au Centre de Recherche en Informatique (CRI) à Fontainebleau et au Centre Energetique et Procédés (CEP) à Sophia Antipolis.

Je souhaite d'abord remercier Robert Mahl, Directeur du CRI et Lucien Wald, Responsable de la formation doctorale, mes Directeurs de thèse pour leurs conseils, leur patience, leur confiance et leur soutien dans la conduite de mes recherches et pour m'avoir accueilli avec les meilleures conditions dans leur centre respectif, de même que Didier Mayer, Directeur du CEP et Thierry Ranchin, Directeur adjoint.

Je tiens aussi à remercier les membres du jury pour l'intérêt qu'ils ont porté à ce travail et plus particulièrement Valérie Roy pour avoir présidé le jury et aussi pour le suivi de mon travail et sa participation active dans la rédaction de ce document. Je remercie Anne Doucet pour avoir soutenu mon travail avec intérêt depuis le début, ses remarques et encouragements m'ont été précieux. Je remercie également Dominique Duportier et Mashan Hacid pour l'aide qu'ils m'ont apporté.

Je remercie sincèrement toute l'équipe du CRI et du CEP qui m'ont aidé tout au long de la thèse afin d'améliorer mes présentations et ma rédaction et notamment le groupe OMD du CEP dans lequel j'ai passé les deux dernières années.

Je remercie spécialement Yves Bertot pour m'avoir orienté et donné un autre point de vue sur mes travaux de recherche.

Je remercie toute l'équipe administrative des deux centres pour leur patience et leur disponibilité.

Je remercie également ma famille pour son soutien.

Table des matières

Glossaire

ARPA	Advanced Research Projects Agency
BPEL4WS, BPEL	Business Process Execution Language for Web Services
BPML	Business Process Modeling Language
CEP	Centre Energétique et Procédés
CORBA	Common Object Request Broker Architecture
CSDL	Composite Service Definition Language
CSIR	Council for Scientific and Industrial Research
DAML-S	maintenant OWL-S
EMP	Ecole des Mines de Paris, maintenant MINES ParisTech
ER	Entité-Relation
ESRA	European Solar Radiation Atlas
GEOSS	Global Earth Observation System of Systems
HTML	Hyper Text Markup Language
HTN	Hierarchical Task Network
HTTP	Hyper Text Transfer Protocol
IARC	International Agency for Research on Cancer
IP	Internet Protocol
MESOR	Management and Exploitation of Solar Resource Knowledge
NCEP	National Centers for Environmental Prediction
NCP	Network Control Program
NOAA	National Oceanic and Atmospheric Administration
OGC	Open Geospatial Consortium
OMM	Organisation Mondiale de la Météorologie
OWL-S	Semantic Markup for Web Services
PDDL	Planning Domain Definition Language
PPM	Polymorphic Process Model
RPC	Remote Procedure Call
SARERD	South Africa Renewable Energy Resource Database
SOAP	Simple Object Access Protocol
SoDa	Solar Databases
SSP	Structural Synthesis of Programme
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UDDI	Universal Description Discovery and Integration
UV	Ultra Violet

W3C	World Wide Web Consortium
XLANG	XML Business Process Language
XML	eXtended Markup Language
XML-RPC	XML Remote Procedure Call
WMO	World Meteorological Organization, cf. OMM
WRDC	World Radiation Data Center
WSFL	Web Service Flow Language
WSCL	Web Service Conversation Language
WSCI	Web Service Choregraphy Interface
WSDL	Web Service Description Language
WSMO	Web Service Modeling Ontology
WSMF	Web Service Modeling Framework

Chapitre 1

Introduction

Pour la science, les données sont nécessaires et leur disponibilité permet de progresser plus vite [118]. Elles sont un moyen d'établir des hypothèses et des modèles bâtis sur ces hypothèses, de vérifier ces hypothèses et de valider ces modèles, puis de les faire évoluer. Dans l'industrie, elles représentent un avantage stratégique, contribuant sur de nombreux plans aux succès d'une entreprise. Par exemple, les données d'ensoleillement sont utilisées pour déterminer la durée de garantie des fenêtres en PVC ou pour déterminer l'impact environnemental d'un viaduc. De nombreuses données sont collectées et stockées par les entreprises et les laboratoires de recherche. Pour exemples, la base de données Hélioclim de l'Ecole des Mines de Paris comporte 24 To de données sur les informations de rayonnement solaire, TerraService comporte 15 To de données aériennes [6] ou encore la NASA collecte 3.5 To de données par jour [116]. Des algorithmes ont été développés pour calculer de nouvelles données à partir de ces données. Par exemple, certains algorithmes peuvent déduire, à partir de quantités de rayonnement global total, des quantités de rayonnement dans l'ultra-violet ; d'autres algorithmes permettent de prédire des vitesses de vent.

Pour accéder à ces données et à ces algorithmes, les chercheurs et les industriels se sont appuyés sur le développement d'Internet et des services Web. Internet est un réseau reliant ces acteurs : fournisseurs de données, fournisseurs d'algorithmes et consommateurs d'informations ; les services Web, quant à eux, fournissent les moyens techniques pour accéder aux données stockées et aux algorithmes transformant ces données.

L'utilisation des services Web n'est pas satisfaisante et il existe en météorologie un grand décalage entre les besoins des utilisateurs et les informations disponibles [24]. Pour utiliser ces informations de manière plus efficace et tirer avantage des services Web déjà disponibles [117, 52], la météorologie doit se doter d'outils permettant de choisir et de combiner les services Web entre eux. La combinaison de ces services Web, de la même façon que le programmeur combine des fonctions entre elles, permet de faire des tâches qu'un service seul n'aurait pas pu réaliser. Cette combinaison s'appelle la composition de services Web.

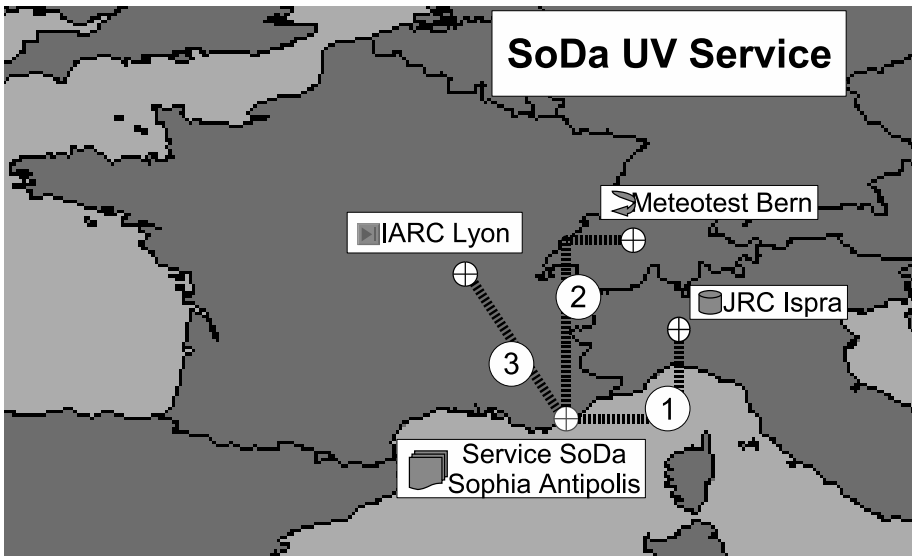

FIG. 1.1 – Illustration d'un cas d'utilisation de l'application SoDa

1.1 Motivation : l'exemple SoDa

Les données et applications en météorologie sont très dispersées géographiquement. Les bureaux météorologiques collaborent depuis très longtemps au sein de l'organisation mondiale de la météorologie (OMM) [71] et travaillent régulièrement à l'élaboration d'outils de plus en plus efficaces afin d'échanger les données et de partager le savoir. Dans les domaines de l'environnement et des énergies, et plus particulièrement pour les aspects liés à la météorologie et l'énergie solaire, des chercheurs et des industriels se sont regroupés dans ce but [24, 32, 97]. Ils ont notamment créé une application appelée SoDa, qui exploite des services Web et rassemble des données et des algorithmes [105]. Elle est largement utilisée par les professionnels, avec 30000 utilisateurs en 2007 [40, 41]. Elle est une illustration d'un outil réalisant la composition de services Web mentionnée ci-dessus.

L'application SoDa est gérée depuis 2003 par l'Ecole des Mines de Paris. Ses gestionnaires ont acquis de l'expérience dans l'exploitation des services Web [40, 41]. Leur expérience, les problèmes qu'ils ont rencontrés, les améliorations qu'ils souhaitaient forment le point de départ de mes travaux. En analysant d'autres applications dans le domaine de la météorologie, j'ai retrouvé des problèmes et des attentes similaires, que j'ai généralisés comme je l'expose plus loin. L'application SoDa est un stéréotype dans la météorologie et elle servira de support dans ma thèse notamment pour tester les solutions proposées.

Pour illustrer le fonctionnement de SoDa et la composition de services Web, prenons le cas illustré par la figure 1.1, qui est celui d'un chercheur du centre de recherche sur le cancer de l'organisation mondiale de la santé à Lyon (IARC). Ce chercheur souhaite connaître l'exposition aux UV des citadins parisiens durant le mois de juin 2005 pour évaluer les risques du cancer de la peau dans cette zone. Pour cela, il peut faire

appel à l'application SoDa.

L'application SoDa dispose, pour ce faire, de deux services Web. Le premier est situé à Ispra en Italie et fournit pour chaque jour une valeur d'irradiation au lieu de coordonnées (x, y) et dans un intervalle de temps D. Il s'agit de l'irradiation totale, c'est-à-dire intégrée sur l'ensemble du spectre. Le second service Web est situé à Berne en Suisse et permet d'extraire l'énergie pour une partie Λ du spectre à partir de l'irradiation totale. Ce service utilise un modèle de distribution du spectre qui nécessite une localisation définie par des coordonnées géographiques (x, y) et une date.

Dans le cas général, l'utilisateur commence par soumettre sa requête, via un navigateur Web, en indiquant le lieu (Paris), l'intervalle de temps (juin 2005) et la partie du spectre lumineux souhaitée (UV). A partir de cette requête, l'application SoDa établit un plan permettant d'y répondre en combinant les services Web. Suivant ce plan, l'application SoDa commence par interroger le service Web situé à Ispra avec le lieu : latitude, longitude, ainsi que l'intervalle de temps (chemin 1 sur la figure 1.1). Le service répond avec les données d'irradiation totale au lieu demandé. L'application SoDa génère alors une nouvelle requête à partir de ces données, de la période (juin 2005), du lieu (Paris) et de la partie du spectre (UV). Cette requête est transmise au service Web situé à Berne (chemin 2 sur la figure 1.1). Ce dernier applique un algorithme pour extraire l'irradiation dans la partie du spectre considérée. L'application SoDa met en forme le résultat et renvoie à l'utilisateur de Lyon les données de rayonnement UV pour la période de juin 2005 (chemin 3 sur la figure 1.1).

Cet exemple montre l'intérêt de composer des services Web car l'utilisateur n'aurait pas pu obtenir les données souhaitées sans combiner les services Web. La composition automatique de services Web s'inscrit dans deux courants : le premier consiste à composer les services Web de façon automatique et statique ; le second consiste à composer les services Web de façon automatique et adaptative. SoDa est un exemple du premier courant.

1.2 Objectifs de la thèse

L'application SoDa prédéfinit un ensemble de compositions possibles de services Web et lie ces compositions avec des requêtes types. Lorsqu'une requête type est fournie, l'application SoDa choisit directement la composition liée à cette requête et l'exécute. Ce type de composition utilisant des compositions prédéfinies et statiques a un domaine de fonctionnement limité. Par exemple, lorsqu'un service Web ne répond plus, toutes les compositions de services Web l'utilisant tombent en panne également. Dans l'exemple précédent, si le service Web situé à Berne tombe en panne, l'utilisateur ne peut plus obtenir les informations souhaitées. Or généralement, il serait possible d'utiliser un service Web remplaçant tout ou partie de la composition en panne et, donc de réaliser cette composition. Par ailleurs, ces compositions étant prédéfinies, l'application SoDa ne peut pas prendre en compte de nouveaux services Web sans l'intervention d'un expert pour redéfinir les compositions et les requêtes types.

Pour mieux répondre aux besoins des chercheurs et des industriels, je souhaite améliorer les méthodes actuelles de composition de services Web. Tout d'abord, selon mes discussions avec eux, les chercheurs et les industriels en météorologie souhaitent préserver ou favoriser l'**automatisation** de la composition. Une composition ne doit pas nécessiter l'intervention de l'homme, en dehors de la soumission de la requête. Par

ailleurs, ils souhaitent que la composition soit **adaptative**, ce qui signifie qu'elle doit prendre en compte l'ajout ou la suppression de services Web ainsi que le résultat de l'exécution de chaque service Web et donc profiter au mieux des ressources disponibles.

L'état de l'art montre qu'il existe deux autres grands types d'approches pour réaliser la composition automatique et adaptative de services Web. Le premier type de composition est basé sur la planification. Il identifie les services Web à des actions qui peuvent atteindre leurs post-conditions quand leurs pré-conditions sont réunies. Il considère la requête comme un but à atteindre. La méthode de planification utilise les actions disponibles pour atteindre ce but. Ce type de méthode, comme le constatent Ponnekanti et Fox avec SWORD [79], peut produire des compositions invalides, comme j'ai pu l'expérimenter lors de mon évaluation personnelle de cette approche. Ce comportement indésirable s'explique par l'absence de la prise en compte du comportement des services Web. Par exemple, un service Web calculant la quantité de rayonnement solaire entre deux heures de la journée peut être utilisé de façon incorrecte parce que l'on ne peut pas différencier l'heure de début de l'heure de fin, alors que cette information influe naturellement sur le résultat, autrement dit, la quantité de rayonnement solaire n'est pas la même entre 7 h et 19 h qu'entre 19 h et 7 h. Un typage plus précis des pré- et post-conditions permettrait de limiter ce type d'erreur mais il compliquerait alors la planification jusqu'à la rendre indécidable.

La seconde approche se base sur la preuve de programme et la génération automatique de preuves ou de programmes. Elle fait l'analogie entre un programme et une preuve de théorème. Chaque service Web est identifié aux axiomes d'une logique donnée et la requête est identifiée à un théorème à prouver. Cette approche utilise ensuite un algorithme dont le but est de trouver une ou plusieurs preuves du théorème et d'en déduire ainsi une composition valide. Bien que ce type de méthode propose toujours des compositions valides vis-à-vis de la requête et des descriptions des services Web, il n'existe pas d'algorithme permettant dans tous les cas de trouver une solution valide, même si celle-ci existe ; l'intervention de l'homme est donc souvent requise. En logique, ce type de problème est dit indécidable.

L'étude bibliographique du chapitre 2 montre qu'il n'existe pas de méthode satisfaisante pour la composition automatique et adaptative de services Web qui prenne en compte les besoins spécifiques de la météorologie. Pour cette raison, mes objectifs sont de proposer, formaliser et développer une méthode qui prenne en compte les améliorations requises présentées dans la section précédente et, de l'appliquer au domaine de la météorologie. Je me suis efforcé de trouver des solutions suffisamment générales pour qu'elles puissent s'appliquer aussi à la climatologie ou encore à de plus vastes domaines comme les géosciences.

1.3 Démarche

Pour atteindre ces objectifs, ma démarche sera la suivante. Je commence, dans le chapitre 2, par analyser les différentes méthodes de composition automatique. Je les ai regroupées en deux catégories ou courants. Je mets en évidence les avantages et les inconvénients de ces catégories vis-à-vis de la composition de services Web grâce à la réalisation de trois prototypes. Je constate en outre dans ce chapitre, l'absence d'un outil d'analyse de la pertinence d'une solution vis-à-vis des améliorations attendues en météorologie.

J'ai donc conçu et réalisé un outil d'analyse de la pertinence d'une solution de

composition de services Web que je présente dans le chapitre 3. Il sert à tester la solution que je propose, mais auparavant, je l'évalue avec les trois prototypes réalisés pour définir les avantages et inconvénients des trois catégories de méthode vis-à-vis des améliorations attendues. Les résultats de cette analyse me guident dans la conception de la méthode que je propose.

J'ai conçu et développé une méthode de composition automatique et adaptative qui emprunte à chacune des trois catégories dans le chapitre 4. Dans la conception, j'ai apporté un grand soin à la description des services Web car le chapitre 2 m'a enseigné l'importance de cette description sur le résultat. La difficulté principale dans le développement est de trouver une méthode qui soit la plus complète possible, répondant le plus possible aux besoins, qui se termine en temps raisonnable et qui soit correcte : il est préférable de n'avoir aucune solution plutôt qu'une solution fausse. Ensuite, je développe la validation de cette nouvelle méthode grâce à l'outil d'analyse de pertinence. On peut ainsi mesurer l'apport de la méthode par rapport à l'existant ainsi que les manques par rapport à une solution idéale.

Le chapitre 5 termine cet ouvrage en tirant les conclusions et en listant quelques perspectives d'étude et d'amélioration.

1.4 Contributions

La première contribution de cette thèse est une formalisation des besoins spécifiques de la météorologie pour la composition de services Web. Cette thèse met en évidence les différences entre les services Web utilisés dans le domaine de la météorologie vis-à-vis des services Web habituellement rencontrés comme les services Web de e-Commerce tels que les services d'achat en ligne de billets d'avion ou de train. Par ailleurs, ma thèse propose également une nouvelle méthode de composition de services Web automatique et adaptative qui est adaptée à la composition de services Web, permettant la concaténation de données et proposant un moyen d'évaluer la qualité de ces compositions. Enfin, je propose une méthode pour évaluer les méthodes de composition, qui répond aux besoins en météorologie.

Chapitre 2

La composition de services Web

2.1 L'Internet et les services Web

L'histoire des services Web commence par la création d'Internet et se poursuit par son formidable essor dans les années 90. L'origine d'Internet est à chercher aux Etats Unis d'Amérique qui souhaitaient se doter d'un réseau de communication capable de résister aux attaques nucléaires. Ce réseau devait pouvoir continer à fonctionner même si une partie venait à être détruite. Les chercheurs vont donc créer un système décentralisé.

En 1964, Paul Baran, un acteur important de la création d'Internet, propose un réseau sous la forme d'une toile. Ce réseau était centralisé, ce qui le rendait vulnérable à des attaques de son système central. Pour l'améliorer, il transforma son réseau en une architecture mélangeant des strutures en étoiles et des structures sous forme de mailles. Les données circulaient dans ce réseau de façon dynamique, empruntant les chemins les moins encombrés et attendant qu'une route se libère. Cette technologie est appelée le *packet switching*.

En août 1969, le réseau ARPANET fut déployé entre quatre universités américaines par Advanced Research Projects Agency (ARPA). Ce réseau n'avait pas d'objectif militaire et il est considéré aujourd'hui comme le précurseur d'Internet. Ce réseau possédait les propriétés essentielles souhaitées par les militaires : le réseau était constitué de noeuds qui pouvaient être détruits sans empêcher le fonctionnement du réseau entre les autres noeuds, les protocoles utilisés étaient rudimentaires. En 1972, ce réseau est présenté au public, et le terme d'*internetting* est utilisé pour désigner ARPANET. Le protocole utilisé était nommé Network Controle Program (NCP) ; il ne gérait pas les erreurs et n'était pas satisfaisant. Ainsi, Alan Kahn développa le protocole TCP. Ce dernier permet de transmettre des données par petits paquets et de prendre en charge les erreurs. Au printemps 1973, il demanda à Vinton Cerf de l'aider à bâtir le protocole.

En 1976, le protocole TCP est déployé sur le réseau ARPANET, qui comportait alors 111 machines reliées entre elles. En 1978, le protocole TCP est partitionné en deux entités distinctes TCP et IP qui vont devenir par la suite TCP/IP. Dès 1980, Tim Berners-Lee, un chercheur du CERN de Genève, mit au point un système de navigation hypertexte et développa, avec l'aide de Robert Cailliau, un logiciel baptisé Enquire permettant de naviguer selon ce principe.

Fin 1990, Tim Berners-Lee met au point le protocole HTTP (Hyper Text Transfer

Protocol), ainsi que le langage HTML (HyperText Markup Language) permettant de naviguer à l'aide de liens hypertextes, à travers les réseaux. Le World Wide Web était né.

Internet a évolué en devenant de plus en plus accessible aux entreprises, aux chercheurs et aux particuliers. Ce réseau est maintenant utilisé par les entreprises pour communiquer avec l'extérieur ou pour relier leurs différents systèmes informatiques. Internet a commencé par être un outil de communication entre des hommes et des machines. Au fil du temps, les entreprises ont relié leurs systèmes d'information de manière à, dans un premier temps, transmettre des informations entre les différentes entités de l'entreprise. Par la suite, les entreprises se sont reliées entre elles pour faire du commerce. Le B2B est né et des infrastructures informatiques comme SAP et des protocoles comme CORBA (Common Object Request Broket Architecture), RPC (Remote Procedure Call) ont été créés pour répondre aux besoins des industriels. Aujourd'hui, beaucoup d'entreprises sont reliées à Internet et un grand nombre d'entreprises et de laboratoires proposent des services Web sur Internet. De plus, beaucoup d'entre elles dépendent du bon fonctionnement d'Internet dans leur vie quotidienne. Ces services Web sont ajoutés, modifiés constamment, d'où le besoin d'outils capables dynamiquement de les utiliser. Pour faciliter la communication entre les entités connectées à Internet, les chercheurs ont proposé trois standards :

- Universal Description Discovery and Integration (UDDI), qui a pour but de répertorier les services Web que proposent les entreprises ;
- Simple Object Access Protocole (SOAP), qui a pour but de favoriser l'interopérabilité en définissant la façon dont doivent communiquer les services Web ;
- et le Web Service Description Language (WSDL) qui permet de décrire le fonctionnement des services Web.

Ces trois standards ont été poussés principalement par IBM et Microsoft. L'UDDI est maintenant principalement utilisé par ces deux contributeurs majeurs. Quant à SOAP et WSDL, ils se démocratisent et sont reconnus comme les standards des services Web. Ca n'a pas toujours été le cas et la définition d'un service Web est sujette à discussions. Les services Web sont parfois appelés e-Services et peuvent être décrits grossièrement comme des ressources appelables par le biais d'Internet. Cette définition conduit à appeler services Web presque toutes les ressources connectées à Internet. Pour cette raison, j'adopterai pour ma thèse la définition proposée par le W3C, qui est plus concrète :

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

(Description des services Web W3C.)

Notons que par abus de langage, on assimile un service Web à une opération de ce service Web. Pour cette raison, dans la suite du document, tous les services Web utilisés ne comportent qu'une seule opération, et j'utiliserai ce terme pour faire référence à cette opération.

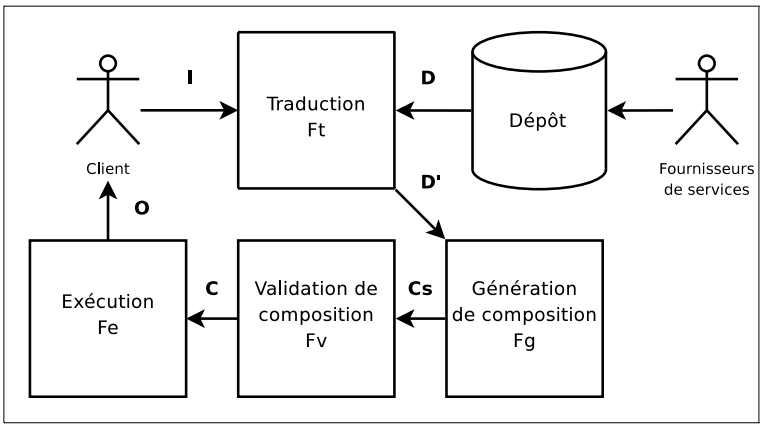

FIG. 2.1 – Schéma de la composition de services Web

2.2 La composition de services Web

La composition de services Web consiste à utiliser plusieurs services Web dans le but de créer de nouveaux services à valeur ajoutée. Comme le montre l'exemple dans l'introduction, combiner des services Web permet, à notre chercheur de Lyon, d'obtenir des données qu'il n'aurait pu avoir autrement. Cette combinaison de services Web, habituellement appelée composition, consiste à combiner plusieurs services Web afin de satisfaire une requête [61, 62, 66, 113]. Elle est un premier pas vers l'accessibilité des informations utiles en tout point géographique à n'importe quel moment en particulier en météorologie [24, 32, 41, 97] mais également dans d'autres domaines [56, 88, 118].

La composition de services Web peut être vue comme une fonction de composition Fc telle que :

$$Fc : I \times D \mapsto O$$

où I est l'ensemble des requêtes possibles de l'utilisateur, O l'ensemble des sorties qu'obtient l'utilisateur et D l'ensemble des ensembles des descriptions de services Web.

Fc est souvent décomposée en sous-fonctions plus simples à réaliser. En m'inspirant de [3, 80], je définis la décomposition suivante, illustrée par la figure 2.1 :

$$Ft : I \times D \mapsto D'$$

$$Fg : D' \mapsto \mathcal{P}(C)$$

$$Fv : \mathcal{P}(C) \mapsto C$$

$$Fe : C \mapsto O$$

Dans cette décomposition, Ft est la fonction de traduction de la description des entrées $i \subset I$ et de l'ensemble des descriptions de services Web $d \subset D$ en une description $d' \subset D'$. D' est l'ensemble des descriptions d' comportant un ensemble de descriptions de services Web et une requête. Le plus souvent, le langage WSDL (Web Service Description Language) [20, 110] est utilisé pour D parce que les services Web

sont souvent décrits par ce langage. Cependant, les descriptions dans ce langage ne sont pas faciles à manipuler pour la fonction de génération de composition Fg. C'est pourquoi la plupart des méthodes de composition définissent une telle fonction Ft qui transforme les descriptions dans le langage WSDL en des descriptions faites dans un autre langage spécifique à chaque méthode. Dans le cas de l'application SoDa, SoDa utilise son propre langage pour la description des services Web et des compositions et les descriptions sont traduites manuellement dans ce langage. Dans ce cas, Ft est réalisée manuellement.

Fg est la fonction qui génère un ensemble de compositions $cs \subset \mathcal{P}(C)$ à partir de la description $d' \subset D'$ de la requête et des services Web. $\mathcal{P}(C)$ est l'ensemble d'ensembles de compositions. Les compositions de cs sont parfois appellées modèles de processus (process model) [74, 83, 98]. Dans le cas de l'application SoDa, cette fonction Fg associe à une requête donnée la composition pré-définie correspondante ; à un d' donné correspond un ensemble de compositions cs ne comprenant qu'un seul élément.

Fv est la fonction qui, à partir d'un ensemble de compositions $cs \in \mathcal{P}(C)$, choisit l'une de ces compositions $c \in C$ suivant différents critères. C est un ensemble de compositions c avec $C \in \mathcal{P}(C)$. Fv peut prendre en compte des attributs non-fonctionnels des services Web, comme la qualité, le domaine d'application, *etc.* [66, 92]. Fv conclut quant à la pertinence des cs de services Web et en choisit une : c. Dans le cas de l'application SoDa, Fv choisit toujours l'unique composition cs donnée par Fg.

Enfin, Fe est la fonction exécutant la composition $c \subset C$ et fournissant la sortie $o \subset O$. Dans le cas de SoDa, comme d'autres applications [17, 52, 117], Fe appelle les services Web dans l'ordre établi par la composition c en liant les variables d'entrées et de sorties entre elles. Dans d'autres applications, Fe réalise des compositions plus complexes comportant, par exemple, des boucles ou des branches de test [13, 35, 109].

La littérature scientifique traite essentiellement des fonctions Fg et Fv. De nombreuses publications leur attribuent une plus grande importance qu'aux fonctions Ft et Fe ; il semblerait que la réalisation de ces dernières ne présentent pas de difficulté scientifique majeure. Dans ce cas, la recherche d'une méthode de composition revient à la recherche d'une méthode de génération : $Fv \circ Fg$, d'une composition $c \subset C$ à partir d'un ensemble de descriptions de services Web et d'une requête $d' \subset D'$. Comme c est l'ordonnancement d'appels à des services Web et que la planification consiste à ordonnancer des actions, la composition de services Web est principalement vue comme un problème de planification comme le montre les citations de ce chapitre.

2.3 La planification : cas général

La planification consiste à établir des plans à partir d'un ensemble d'actions pour effectuer une tâche plus élaborée [87]. Le plan est une représentation de l'organisation des actions dans l'espace et le temps. Par exemple : "utiliser le service Web situé à Ispra puis utiliser le service Web situé à Bern" est un plan. Il existe de nombreuses applications à la planification comme celles dont l'objectif est de trouver un chemin, celles pour planifier des mouvements, ou organiser des agents autonomes [57, 106]. Je m'intéresse à l'ordonnancement d'actions qui seront réalisées par des services Web.

Un problème de planification peut être représenté comme un système à états [4, 9, 80, 91]. Carman *et al.*[16] le définit à l'aide du quintuplé $\langle \mathbf{S}, s_i, s_f, \mathbf{A}, \mathbf{R} \rangle$. $\mathbf{S}$ est l'ensemble des états possibles du système considéré, s_i est l'état initial du système

et s_f est l'état du système que l'on souhaite obtenir. s_i et s_f appartiennent à $\mathbf{S}$. $\mathbf{A}$ est l'ensemble des actions possibles. Chaque action est définie par un ensemble de pré-conditions et de post-conditions sur les états, ces dernières étant souvent appelées effets. Les pré-conditions doivent être réalisées pour que l'action puisse être appliquée et les post-conditions sont forcément réalisées après une action. $\mathbf{R}$ est l'ensemble des transitions d'état possibles ; $\mathbf{R}$ est donc inclus dans $\mathbf{S} \times \mathbf{A} \times \mathbf{S}$ et représente les applications possibles des actions à chaque état donné $s \in \mathbf{S}$ ainsi que leurs conséquences : le nouvel état produit. Résoudre le problème de planification revient à trouver un enchaînement d'actions permettant de relier l'état s_i à l'état s_f.

Le lien entre la composition de services Web et la planification est immédiat dès lors que les objets manipulés dans le plan sont remplacés par des données et les actions par des services Web manipulant ces données.

Pour résoudre un problème de planification, les conditions suivantes doivent être réunies :

- le nombre d'états s de $\mathbf{S}$ doit être fini ;
- le nombre des actions a de $\mathbf{A}$ doit être fini ;
- S décrit tous les états possibles du système, *i.e.* la connaissance des états est complète ;
- les actions sont déterministes, *i.e.* une action ne doit pas avoir d'effets aléatoires ;
- les actions sont instantanées, et sont indépendantes, *i.e.* elles ne sont pas coordonnées.

Dans ces conditions, il est possible de trouver un plan permettant d'atteindre s_f. Or, pour la composition de services Web, ces conditions ne peuvent pas toujours être réunies. Par exemple, si un système manipule des entiers, ceux-ci étant en nombre infini, il est imposible de représenter tous les états possibles.

Par ailleurs, la planification ne cherche habituellement qu'un plan possible, alors que dans la composition de services Web la fonction Fg doit en fournir plusieurs. L'idéal serait que Fg fournisse toutes les compositions possibles ; or même si la planification peut en fournir une, elle ne peut généralement pas les fournir toutes. Enfin, si on limite le nombre de plans, il n'est pas possible de savoir s'il n'existe pas un autre plan meilleur.

Comme nous l'avons indiqué, la composition de services Web est principalement vue comme un problème de planification. Cette planification s'incarne dans deux courants de composition [33]. Le premier courant est basé sur la planification statique alors que le second est basé sur la planification dynamique.

2.4 Planification statique pour la composition de services Web

La planification statique consiste à pré-résoudre les problèmes de planification. Ceux-ci sont définis en fonction des besoins de l'utilisateur : si nous savons que de nombreux utilisateurs auront besoin de données de rayonnement solaire, alors nous résolvons à l'avance ce type de requête. Les plans sont donc pré-définis en fonction d'une ou plusieurs requêtes type. Ce type d'application est celui qui est aujourd'hui le plus utilisé, en particulier, par les industriels. Il existe deux visions de la composition statique qui sont l'orchestration et la chorégraphie [75, 77, 94]. L'orchestration [21, 103] aborde le problème de façon centralisée, où les compositions de services Web statiques sont réalisées par un composant qui se charge d'ordonner les appels

aux services Web et de rattraper les erreurs. En parallèle, la chorégraphie [7] aborde le problème de façon distribuée, chaque partenaire d'une composition, *i.e.* chaque fournisseur de services Web, peut réaliser une ou plusieurs tâches, chacun d'eux communicant à l'aide de services Web. Ce type de composition statique s'appuie sur des langages de composition de services Web tels que :

- XLANG (XML Business Process Language) de Microsoft ;
- BPML (Business Process Modeling Language) de BPMI ;
- WSFL (Web Service Flow Language) de IBM ;
- WSCL (Web Service Conversation Language) de Hewlett-Packard ;
- WSCI (Web Service Choregraphy Interface) de SUN [109] ;
- BPEL4WS (Business Process Execution Language for Web Services) de l'association de IBM, Microsoft et BEA [50, 99, 102], aussi appelé BPEL ou WSBPEL.

Ces langages décrivent les interactions entre différents fournisseurs de services Web et leurs clients. XLANG, BPML et BPEL sont associés à l'orchestration alors que WSCL et WSCI sont associés à la chorégraphie. Par ailleurs, il existe quelques autres méthodes dans la litérature, j'en présente une liste qui se veut la plus exhaustive possible.

SoDa [40, 104] est une méthode utilisant un modèle de plan statique. Chaque composition est définie comme une séquence de services Web à interroger. SoDa ne possède pas de structure de composition avancée de type disjonction, conjonction, parallélisation, ce qui limite beaucoup ses capacités. De plus, il définit une tâche comme l'appel d'un service Web particulier, autrement dit, il ne sélectionne pas nécessairement les services Web les plus appropriés.

EFlow [17, 19] est une plate-forme pour la spécification, l'établissement et le management des services Web composés. EFlow utilise un générateur de modèles de plan statique. Un service Web composé est modélisé par un graphe qui définit l'ordre d'exécution des tâches ; celles-ci sont représentées par des noeuds dans le graphe et peuvent être réalisées par des services Web. Le graphe modélisant un service Web composé et la base de données des services Web sont créés manuellement. Le graphe peut contenir des noeuds de services, de décisions et d'événements. Les noeuds de services représentent l'appel à un service Web, élémentaire ou composé, les noeuds de décisions spécifient les alternatives et les règles contrôlant l'exécution et le flux de données, par exemple une disjonction ou une jonction. Enfin, les noeuds d'événements permettent aux services Web de recevoir et d'envoyer un certain nombre de signaux durant l'exécution. Les arcs du graphe montrent les dépendances entre les noeuds, *i.e.* noeuds de services, de décisions ou d'évènements. Lors de l'appel d'un service Web composé, EFlow sélectionne automatiquement les services Web correspondant aux noeuds de services. Lors de la définition d'un service Web composé, l'utilisateur peut rechercher et définir les services Web qui seront utilisés pour chaque noeud de services [22]. EFlow propose également des procédures de modification des modèles de plan de manière à répondre au mieux aux besoins des utilisateurs. Dans ce cas, lorsqu'un noeud de services est appelé, une recherche de services est entreprise et celle-ci fournit des références vers les services Web disponibles. De manière générale, la recherche est appelée à chaque exécution d'un noeud de service dans le graphe car la disponibilité des services Web peut changer souvent.

Dans [18], les auteurs redéfinissent la plate-forme de composition de services et proposent un prototype de langage de définition de services composés (Composite Service Definition Language : CSDL). Une intéressante fonctionnalité de CSDL est la distinction entre les services et les opérations des services. Les auteurs definissent donc

les noeuds de services et des noeuds d'opérations, les premiers faisant référence aux services, les seconds s'adressant directement à une méthode particulière d'un service. Selon les auteurs, CSDL fournit les fonctionnalités adaptatives et dynamiques qui correspondent à l'évolution rapide des entreprises et des environnements informatiques et technologiques dans lesquels les services Web sont utilisés.

Polymorphic Process Model (PPM) [89] utilise une méthode qui combine le modèle de plan statique avec le modèle de plan dynamique. Les auteurs décrivent la composition de services Web à l'aide d'activités. PPM comporte deux type d'activités, les activités génériques, proposées et exécutées par le moteur de PPM et les activités spécifiques à l'application qui doivent être définies par l'utilisateur. PPM choisit de séparer les implémentations de ces activités, réalisées par les services Web, et les interfaces de ces activités, qui sont une représentation des services Web. Les interfaces des activités sont modélisées comme des machines à états, qui incluent des états et des opérations permettant la transition d'états, et sont également décrites par leurs entrées / sorties. Les activités spécifiques à l'application sont des abstractions du comportement des services Web. La réalisation des activités par PPM est effectuée en liant les opérations des activités avec des opérations de services Web spécifiques. Cette tâche est similaire à la liaison des services Web dans EFlow, pour lequel les implantations des interfaces sont liées lors de l'exécution.

2.5 Planification dynamique pour la composition de services Web

Le second courant est celui de la composition dynamique basée sur la planification dynamique principalement. La composition dynamique, que j'appelle composition automatique et adaptative, est différente de la composition statique car elle propose de chercher une composition de services Web au moment de la requête de l'utilisateur. Elle n'utilise donc pas de plans prédéfinis. Ce type de composition est encore très peu utilisé car il n'est pas encore assez sûr, c'est-à-dire que l'obtention et la qualité du résultat ne sont pas garanties. Le W3C propose une méthode de description des services Web visant à faciliter la composition de services Web automatique et adaptative. Cette description se base sur un langage appelé OWL-S [74] pour Ontology Web Language - Semantics, issu du langage DAML-S [25]. Ce langage permet de décrire les services Web de façon à pouvoir les utiliser plus efficacement. Il existe également un autre langage appelé WSMO [111] dont l'objectif est similaire. Ce langage, issu du WSMF (Web Service Modeling Framework) [35] se base sur les ontologies. Il existe des différences entre ces deux langages de description rapportées par [55], mais elles ne semblent pas fondamentales. J'ai choisi de présenter OWL-S car il est plus mature [55] et je m'inspirerai des informations qu'il fournit pour établir ma méthode de composition.

OWL-S : les services Web sémantiques OWL-S est un langage défini avec les objectifs suivants. Son premier objectif est de permettre la découverte automatique des services Web. Celle-ci consiste à rechercher les services Web qui répondent à une classe particulière de problème tout en répondant à des contraintes spécifiques du client. Par exemple, l'utilisateur peut souhaiter trouver un service Web fournissant des quantités de rayonnement solaire. Trouver ce service Web est encore aujourd'hui essentiellement manuel. L'utilisateur peut s'appuyer sur des annuaires répertoriant

les services Web tels que les annuaires UDDI [101]. Ce type d'annuaire propose des moteurs de recherche de services Web, les utilisateurs peuvent les utiliser puis, ensuite, ils peuvent choisir un service Web en fonction de ses contraintes en lisant la description des différents services Web proposés. OWL-S propose donc un moyen de découvrir plus facilement les services Web.

L'objectif suivant de ce langage est de fournir un moyen pour interroger automatiquement les services Web, c'est-à-dire qu'un programme informatique qui n'a pas été programmé spécifiquement doit être capable d'interroger ce service Web en lisant les informations fournies par la description en OWL-S. Par ailleurs, un point important de OWL-S est de permettre la composition automatique des services Web. Pour y parvenir, OWL-S propose une description détaillée comportant des informations utiles en ce but.

OWL-S fournit trois connaissances essentielles sur les services Web, qui répondent aux questions suivantes :

- que fournit le service Web pour l'utilisateur potentiel ? La réponse à cette question est fournie par le *ServiceProfile*,
- comment fonctionne le service Web ? La réponse à cette question est fournie par le *ServiceModel*,
- comment le client peut-il interagir avec le service Web ? La réponse à cette question est fournie par le *ServiceGrounding*.

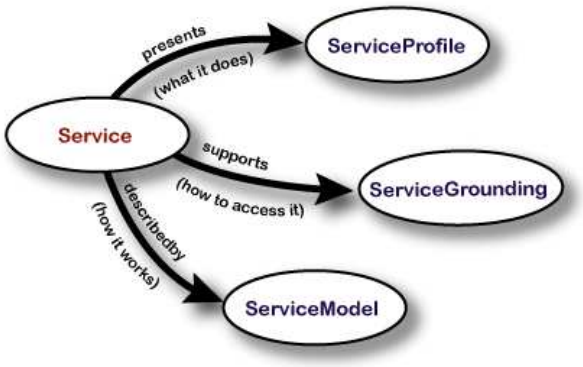

FIG. 2.2 – Niveau supérieur de l'ontologie de OWL-S [74]

OWL-S [74, 96] décrit donc les services Web suivant trois points de vue différents qui sont le *ServiceProfile*, le *ServiceModel* et le *ServiceGrounding* représentés dans la figure 2.2. Le *ServiceProfile* fournit des informations sur le domaine d'application du service Web. Chaque service Web est rangé dans une taxinomie facilitant la recherche du service Web pour les clients potentiels [5]. Le *ServiceProfile* fournit également des éléments sur les conditions d'exécution du service telles que les préconditions et les effets du service Web au sens de OWL-S, ainsi que la description des entrées/sorties. L'ensemble de ces conditions d'exécution est proche de l'ensemble des pré-conditions et post-conditions de la planification. De plus, ces informations sont liées au *ServiceModel*. Ce dernier décrit les fonctionnements des services Web en les décomposant en *AtomicProcess*, *SimpleProcess* ou *CompositeProcess*. Les *AtomicProcess* correspondent à des actions qu'un service Web peut effectuer. Les *AtomicProcess* ne comportent pas de sous-processus et n'exécutent qu'une seule étape consistant à

interroger un service Web. Ils prennent un message d'entrée puis font quelquechose et renvoient un message. Pour chaque *AtomicProcess*, le service Web correspondant doit être défini dans le *ServiceGrounding*. Les *SimpleProcess* ne peuvent pas être appelés directement comme les *AtomicProcess* et ne sont pas associés à des services Web dans le *ServiceGrounding*. En revanche, ils sont vus comme des procédures ne comportant qu'une seule étape et pouvant être réalisées par un *AtomicProcess* ou un *CompositeProcess*. Ils sont en fait une vue alternative des deux autres procédures. Enfin les *CompositeProcess* sont des procédures complexes regroupant plusieurs étapes pour être réalisées. Elles sont décomposables en *AtomicProcess*, *SimpleProcess* et *CompositeProcess*. Cette décomposition peut être spécifiée à l'aide de structures de contrôle telle que *if-then-else* ou les *sequence*. Les structures de contrôle proposées par OWL-S sont :

- *Sequence* : la séquence est une liste de processus qui doivent être exécutés dans l'ordre ;
- *Split* : appelle des sous-processus de façon concurrente ;
- *Split+join* : consiste à lancer différents processus de façon concurrente mais de les synchroniser en attendant que toutes les sous-procédures du split+join soient achevées pour les joindre ;
- *Any-order* : définit l'exécution d'un ensemble de tâches dans un ordre quelconque ;
- *Choice* : le choix consiste à choisir entre plusieurs sous-procédures équivalentes ;
- *If-Then-Else* : correspond au *if-then-else* habituel, c'est-à-dire si la condition du *if* est réalisée, alors la sous-procédure du *then* est exécutée, sinon la sous-procédure du *else* est exécutée ;
- *Iterate* : boucle continuellement avant d'être interrompu par *whileCondition* ou *untilCondition* ;
- *Repeat-While* et *Repeat-Until* : sont des boucles conditionnelles qui répètent une sous-procédure tant que (*Repeat-While*) la condition de la boucle n'est pas satisfaite ou jusqu'à ce que (*Repeat-Until*) la condition de la boucle soit réalisée.

Enfin le *ServiceGrounding* définit les entrées et les sorties du service Web ainsi que les informations nécessaires pour appeler le service Web. Il reprend en partie les informations fournies par le WSDL comme le montre la figure 2.3. Il permet de lier les *AtomicProcess* avec des services Web existants.

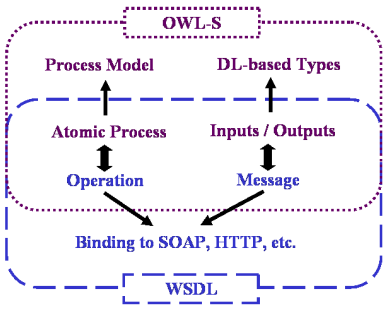

FIG. 2.3 – Lien entre le wsdl et OWL-S [74]

En parallèle au développement de WSMO et de OWL-S se développent des mé-

thodes de composition dynamique. Cette section en présente quelques-unes. Ces méthodes sont issues généralement de méthodes de planification existantes et de méthodes de preuve automatique de programme.

McIlraith et *al.* [64, 65] adaptent et étendent le langage Golog pour la construction de services Web composés. Golog est un langage de programmation logique construit sur le calcul de situation. Ce dernier consiste à modéliser l'état du monde comme un arbre de situations, démarrant d'une situation initiale S_0 et évoluant vers de nouvelles situations par l'application de différentes actions a. Les auteurs traitent le problème de composition de services Web en considérant ceux-ci comme des actions complexes. Ils considèrent que Golog est un formalisme naturel pour la représentation et le raisonnement vis-à-vis des services Web.

L'idée générale de cette méthode est que les agents informatiques pourraient raisonner sur les services Web pour les découvrir, les exécuter et les composer automatiquement. La requête de l'utilisateur et les contraintes peuvent être écrites dans le langage de la logique du premier ordre utilisé pour le calcul de situation. Les auteurs conçoivent tous les services Web comme des actions – les actions primitives et les actions complexes. Les actions primitives sont conçues comme à la fois des actions qui changent l'état du monde ou des actions délivrant de l'information qui changent l'état des connaissances de l'agent. Les actions complexes sont des compositions d'actions primitives. Les bases de connaissance des agents fournissent un format de codage logique des pré-conditions et des post-conditions (effets) des actions dans le langage du calcul de situation. Les auteurs définissent des modèles de programmes. Ces modèles de programmes, écrits en ConGolog [39], définissent ce qu'ils réalisent plutôt que comment ils doivent le réaliser. Le langage utilisé par les auteurs pour décrire ces modèles combine des opérations des langages de programmation procédurale, comme la séquence, si-alors-sinon ou le tant que, avec des opérateurs définis dans DAML qui sont spécifiques aux services Web. De plus, le langage permet de définir des contraintes basées sur des ontologies.

Les auteurs proposent aussi une manière de personnaliser les programmes Golog par l'incorporation de contraintes par l'utilisateur. Par exemple, l'utilisateur d'un service Web composé peut utiliser le choix non-déterministe, c'est-à-dire un choix aléatoire, pour présenter les actions sélectionnées dans une situation donnée, ou utiliser le concept de séquence pour forcer l'ordre d'exécution entre deux actions. La génération du plan doit obéir aux contraintes prédéfinies.

Le grand intérêt de la communauté de la composition de services Web pour la planification peut être expliqué simplement par les similarités entre les représentations du langage DAML-S et du langage PDDL [38] (Planning Domain Definition Language). Ce dernier est un langage standard qui unifie la représentation des actions. Il est utilisé pour comparer les différentes méthodes de planification [87]. Le DAML-S [69, 90] s'est fortement inspiré du langage PDDL [74]. Grâce à ses affinités avec le PDDL, le DAML-S bénéficie de plusieurs planificateurs ou générateurs de plan.

Dans la représentation de la méthode de composition de services Web basée sur le PDDL, [63] introduit un nouveau type de connaissance pour chaque action appelée *value*. Cette connaissance persiste et n'est pas traitée comme une clause de vérité. Du point de vue de la construction de services Web composés, la fonctionnalité permet de distinguer la transformation ou l'acquisition des informations d'une part et le changement d'état d'autre part, qui sont produits par l'exécution d'un service Web. L'information représentée par les paramètres d'entrées/sorties, est supposée être réutilisable. Ainsi, les valeurs des données fournies par les services Web peuvent être dupliquées

pour l'exécution de plusieurs services Web. De manière résumée, l'exécution d'un service Web peut changer l'état du monde, impliquant la disparition de l'ancien état et la création d'un nouvel état, et peut fournir des connaissances *value* persistantes et réutilisables tout au long de la composition.

La notion d'information persistante est importante pour la composition de services Web. Habituellement dans la planification, on fait l'hypothèse d'un monde clos, qui suppose que, si une formule n'est pas explicitement définie comme vraie dans un état, alors elle peut être supposée fausse dans cet état. On parle de monde clos par opposition au monde ouvert dans lequel on considère qu'une croyance qui n'existe pas, n'est pas nécessairement fausse. Dans la programmation logique, on appelle l'hypothèse du monde clos "negation as failure", soit un échec est interprété comme une négation. McDermott [63] remarque que le principal problème avec l'hypothèse du monde clos, du point de vue du concept des services Web, est que le planificateur connaît souvent l'existence de valeurs dont il ne connaît pas la valeur. Par exemple, le planificateur sait qu'il existe une valeur d'éclairement dont il ne connaît pas la valeur, ce qui est en contradiction avec l'hypothèse du monde clos. McDermott contourne le problème en différenciant des valeurs *apprenables*, par exemple "apprenable éclairement" (learnable *value*), et des valeurs *connues*, par exemple "connue éclairement" (know *value*).

Medjahed et *al.* [66] présentent une technique pour générer une composition de services à partir d'une description de haut niveau. Ils étendent le langage WSDL comme le propose également [28]. La méthode utilise des règles de composabilité pour déterminer si deux services sont composables. Elle se décompose en quatre phases. La première, la phase de spécification, permet un haut niveau de description de la composition décrite en utilisant un langage de spécification de services composés appelé CSSL (Composite Service Specification Language). La seconde, la phase d'appariement, utilise des règles de composabilité pour générer des plans de composition qui sont conformes aux spécifications de la requête de l'utilisateur. La troisième est la phase de sélection. Si plus d'un plan est généré durant la seconde phase, l'utilisateur du service choisit un plan en fonction de différents paramètres tels qu'un paramètre de qualité de composition, son rang ou son coût. La phase finale est la phase de génération. Une description détaillée de la composition des services Web est générée automatiquement et est présentée à l'utilisateur du service.

Ce paragraphe présente plus en détails les règles de composition afin de mieux exposer comment est généré le plan. Les règles de composabilité considèrent des propriétés syntaxiques et sémantiques des services Web. Les règles syntaxiques incluent les règles pour les modes de fonctionnement, par exemple l'envoi de message uniquement (one-way) ou l'envoi de requête puis l'attente d'une réponse (request-response), et les règles de liaison des protocoles d'interaction des services Web. Les règles sémantiques incluent les sous-ensembles suivants :

- *composabilité des messages définis* : deux services Web sont composables si et seulement si le message de sortie d'un service est compatible aux sens des auteurs avec le message d'entrée de l'autre service,
- *composabilité sémantique de l'opération* : elle définit la compatibilité entre les domaines, les catégories et les propositions (fonctionnalités) de deux services,
- *qualité de composition* : elle définit les préférences de l'utilisateur vis-à-vis de la qualité des opérations pour la composition de services,
- *pertinence de la composition* : elle évalue si une composition de services est bien fondée.

Enfin, les auteurs introduisent la notion de modèles de composition qui définissent les

dépendances entre différents services Web.

Selon moi, la contribution principale de cette méthode est la définition des règles de composabilité, car elles montrent qu'elles peuvent être les attributs des services Web utilisables pour la composition automatique et adaptative de services Web. Ces règles peuvent être utilisées comme guide pour les autres méthodes basées sur la planification.

SWORD [79] est un autre kit de développement pour construire des services Web composés utilisant des règles pour la génération de plan. Contrairement à [66], SWORD n'utilise pas les standards de description émergents tels que [110] ou [74], mais exploite à la place, le modèle entité-relation (ER) [36]. Dans SWORD, les services Web sont modélisés par leurs pré-conditions et leurs post-conditions. Les pré-conditions et post-conditions sont spécifiées dans un modèle qui consiste en un ensemble d'entités, par exemple une voiture, une personne, un courrier, et des relations entre les entités, par exemple X possède Y. Un service Web est représenté sous la forme de clauses de Horn. Une clause de Horn est une disjonction de littérales dont au plus l'une d'elles est positive. Ces clauses montrent que les post-conditions sont réalisées si les pré-conditions sont vraies. Ces clauses de Horn [46] traduisent simplement une implication ; la clause de Horn :

$$\neg p \vee \neg q \vee \ldots \vee \neg t \vee u$$

traduit l'implication :

$$p \wedge q \wedge \ldots \wedge t \rightarrow u$$

où $\vee$ est le "ou" logique, $\neg$ est la négation et $\wedge$ est le "et" logique. Pour créer un service Web composé, il suffit de spécifier l'état initial et l'état final du service Web composé, puis la génération de plan peut être réalisée en utilisant un système expert à base de règles.

Dans le domaine des méthodes de composition, un intéressant travail effectué dans SWORD est que les auteurs discutent la composition de services Web basée sur des règles, et montrent qu'elle peut parfois générer un résultat incertain si les pré-conditions ne peuvent pas déterminer de façon unique les post-conditions (effets). Par exemple, prenons deux services Web S_1 et S_2 : S_1 fournissant $b(X)$ à partir de $a(X)$ et S_2 fournissant $c(X)$ à partir de $b(X)$. Si l'utilisateur désire obtenir $c(X)$ en fournissant $a(X)$ alors SWORD proposera d'enchaîner les services Web S_1 puis S_2. Maintenant si le service S_1 fournit l'adresse personnelle de quelqu'un à partir de son nom, que S_2 fournit le numéro de téléphone à partir d'une adresse donnée et que deux personnes peuvent vivre à la même adresse en ayant chacune un numéro de téléphone, alors le résultat de la composition est incertain. Les auteurs prétendent que les résultats incertains peuvent être éliminés seulement si les pré-conditions sont fonctionnellement dépendantes des post-conditions dans les services Web utilisés.

Quelques autres techniques de planification sont proposées pour la composition automatique. Dans [113], le planificateur de SHOP2 [53, 93] est utilisé pour la composition automatique de services Web qui sont décrits grâce au langage DAML-S. SHOP2 est un planificateur de type réseau de tâches hiérarchiques (HTN, hierarchical task network). Les auteurs pensent que le concept de décomposition de tâches dans le planificateur HTN est très similaire au concept de décomposition de processus composés, décrits dans l'ontologie de processus proposée par DAML-S. Les auteurs avancent aussi que le planificateur HTN est plus efficace que les autres langages de planification, comme par exemple Golog. Dans leurs publications, les auteurs

donnent une description détaillée du processus de traduction Ft du langage DAML-S vers le langage utilisé dans SHOP2, en particulier pour les structures de contrôle (par exemple : si-alors-sinon, séparation, jointure) qui peuvent pour la plupart être traduites de manière explicite en SHOP2.

Sirin et *al.* [92] présentent une méthode semi-automatique pour la composition. La composition semi-automatique consiste à assister l'utilisateur lors de la création d'une composition. En fait, la composition est assistée par ordinateur. Sirin et *al.* proposent une méthode qui utilise DAML-S. La méthode proposée utilise les attributs fonctionnels et des attributs non-fonctionnels définis dans le *ServiceProfile* pour proposer à l'utilisateur les services qui semblent les plus appropriés pour répondre à sa requête. Pour ce faire, les attributs fonctionnels comme paramètres d'entrées et de sorties du service sont représentés par des classes OWL [73] et sont filtrés par un moteur d'inférence basé sur OWL et Prolog. Le moteur d'inférence peut ordonner les services Web filtrés en fonction de l'ordre d'éloignement des concepts. L'éloignement est un paramètre qui dénote l'importance des différences entre les classes OWL. Pour affiner le résultat, si plus d'une correspondance est trouvée, le système filtre le résultat en fonction des contraintes fournies par l'utilisateur sur les attributs non-fonctionnels. Les attributs non-fonctionnels sont les attributs utiles qui ne sont pas fournis par les attributs fonctionnels, par exemple, la localisation de la mesure lorsque le service ne la fournit pas lui-même. Seuls les services Web qui passent le filtre de contraintes sont présentés à l'utilisateur. La composition semi-automatique est attrayante car elle permet de surmonter les difficultés liées, tout d'abord, à la capture des besoins de l'utilisateur, puis à la composition automatique qui, le plus souvent, ne permet pas de garantir l'exactitude de la composition. L'utilisateur étant pro-actif, il peut s'assurer que la composition est bien celle qu'il souhaitait. De plus, la méthode proposée est simple et montre que la génération de services Web composés peut être effectuée en combinant les fonctionnalités de la machine et les compétences de l'homme.

Enfin il existe des méthodes basées sur la preuve automatique.

Waldinger [106] propose de générer les services Web composés à partir de preuves de théorèmes. L'approche est basée sur la déduction automatique de preuves. La requête de l'utilisateur est décrite comme un théorème que l'on souhaite prouver. Initialement, les services Web disponibles et les pré-requis de l'utilisateur sont décrits dans la logique du premier ordre. L'utilisateur définit des axiomes de la logique classique, par exemple, des implications ou des équivalences, qui vont être utilisées pour générer une ou plusieurs preuves de la requête. Ensuite, une preuve est générée par le générateur de preuve de théorème SNARK. Enfin, la description de la composition de services est extraite d'une preuve particulière en fonction de la disponibilité des services Web.

Lämmermann [54] applique une synthèse structurelle de programme (SSP, Structural Synthesis of Programme) pour une composition automatique. SSP est une approche déductive pour la synthèse de programme à partir de spécifications. Il définit un langage logique dans lequel les spécifications sont définies par des interfaces. Les interfaces définissent les services Web ou la requête. Ces interfaces sont composées de variables typées, de constantes, de liaisons entre les variables et constantes (binding), d'un axiome. L'axiome inclut seulement les propriétés structurales, i.e. les informations des entrées/sorties, en liant les entrées avec les sorties. Il étend son langage en y ajoutant des "variables de contrôle" qui sont des variables sans valeur mais qui peuvent servir à la définition des pré-/post-conditions. Les interfaces des services Web sont vues comme des axiomes et l'interface de la requête est vue comme un

théorème à prouver. SSP déduit des interfaces prédéfinies une "preuve" de l'interface qui représente la requête, et déduit de la preuve la composition des services Web. Lämmermann assimile les services Web composés à des programmes et utilise le fait que les programmes sont des preuves.

Rao et *al.* [80, 81, 82] introduisent une méthode pour la composition automatique de services Web sémantiques utilisant des preuves de théorème en logique linéaire (linear logic theorem proving). Cette méthode utilise deux représentations pour les services Web. La première est une représentation externe utilisant le langage OWL-S. La seconde représente les services Web par des axiomes logiques dans la logique linéaire, comme le fait Waldinger [106] dans la logique classique. La logique linéaire permet à l'utilisateur de définir les attributs des services Web formellement, incluant qualification et quantification des valeurs des attributs non-fonctionnels.

De plus, la logique linéaire a des relations avec le pi-calcul. Le pi-calcul est la fondation formelle de certains langages de composition de services Web [1, 13, 15, 43, 59]. La possibilité de considérer la preuve de logique linéaire comme un processus de pi-calcul a été pour la première fois montrée par [2], et ensuite développée par [8]. Les auteurs attachent le pi-calcul aux règles d'inférence de la logique linéaire dans le style de la théorie des types. Ainsi, le modèle de procédure pour la composition de services Web représenté par le pi-calcul peut être généré directement à partir de la preuve. Les auteurs présentent aussi les règles de sous-typage qui sont utilisées pour raisonner avec les figures d'inférence de la logique linéaire. Ainsi, le générateur de preuve de théorème de logique linéaire peut négocier avec la spécification du service et l'information du Web sémantique. Contrairement à d'autres méthodes, comme [92, 66] qui utilisent les attributs non-fonctionnels seulement pour filtrer les plans générés, Rao *et al.* considèrent les attributs non-fonctionnels directement dans le processus de preuve de théorème, *i.e.* durant la génération de plans. Les fonctionnalités des services et les attributs non-fonctionnels sont traduits par des axiomes logiques. La distinction entre les fonctionnalités et les attributs non-fonctionnels est permise par les règles d'inférence de la logique linéaire.

L'utilisation des méthodes de composition basées sur la preuve de programme est nuancée par le fait que la recherche de preuve dans la logique linéaire est indécidable [58]. Ceci implique que dans certaines situations, les preuves ne peuvent être trouvées, soit parce que c'est trop long d'en trouver une, auquel cas on arrête le processus de recherche, soit parce que la preuve n'existe pas, auquel cas les méthodes de preuve automatique ne pourront pas s'arrêter pour conclure qu'il n'existe pas de preuve.

2.6 Bilan

Nous avons classé les méthodes de composition que nous avons trouvées dans la littérature en trois groupes : statiques, dynamiques et inductives. Dans le chapitre suivant nous abordons le cas particulier de la météorologie et nous en formalisons les besoins pour la composition. Au chapitre 4 nous ferons une étude préliminaire de ces trois groupes à l'aide de prototype en prenant en compte le point de vue de la météorologie.

Chapitre 3

Application à la météorologie

Dans ce chapitre, je décris succinctement la météorologie telle qu'elle est étudiée dans le centre énergétique et procédés (CEP) de MINES ParisTech. J'expose les besoins exprimés par cette équipe, les utilisateurs du service SoDa ou d'autres services analogues, sur la composition de services Web. Ensuite, je présente les particularités des services Web en météorologie et je montre que ce domaine tirerait un grand bénéfice de la composition. J'exprime les contraintes liées à ce domaine et leurs impacts sur la composition. L'expression des besoins des utilisateurs et l'étude des caractéristiques intrinsèques aux services Web liés à la météorologie me permettent de formaliser mathématiquement les compositions que souhaitent réaliser les utilisateurs des services SoDa et autres, et donc de préciser les objectifs de la thèse.

3.1 La météorologie

La météorologie couvre des aspects d'observation, de modélisation et de décision. Dans le domaine de l'énergie, en particulier, les chercheurs y font appel pour le développement de méthodes et d'outils de représentation mathématique de la réalité géographique, en la combinant avec d'autres disciplines scientifiques comme la géographie numérique, les mathématiques appliquées à la science de l'information ou la métrologie. La météorologie contribue notamment à l'accroissement de l'usage des énergies renouvelables pour la production d'énergie. Compte tenu du contexte mondial climatique et écologique comme le réchauffement de l'atmosphère ou encore l'épuisement des réserves naturelles, ces préoccupations sont désormais sur le devant de la scène politique internationale. Cette situation rend d'autant plus importantes les innovations technologiques pour l'amélioration de l'accessibilité des données météorologiques, comme par exemple l'utilisation d'Internet et des services Web pour la diffusion de ces données et des méthodes pour les exploiter.

Les informations manipulées en météorologie ont des origines très diverses : réseaux de capteurs terrestres, satellites d'observation de la terre, modèles numériques. Des travaux importants sont donc menés continuellement pour la fusion d'informations. L'ingéniérie des énergies renouvelables requiert également une grande quantité d'information d'autres natures, comme par exemple, dans le cas d'une implantation de centrale solaire thermodynamique, le relief du terrain et les réseaux hydrographique, électrique et des voies de communication. Rassembler toutes ces informations en un

même lieu ne constitue plus la solution idéale et on observe, comme pour d'autres sciences liées à l'environnement, le développement de solutions d'interopérabilité, incluant les services web et leur composition.

3.2 Les systèmes d'information disponibles en météorologie

Les météorologues ont développé un réseau de partage de données qui leur est propre. Ce réseau est basé sur les différents bureaux de météorologie de différents pays. Ces derniers proposent leurs données sous diverses formes et à des coûts variables. Par exemple, le gouvernement américain propose ces données gratuitement pour ses citoyens car il considère que le prix d'acquisition des données a déjà été payé par les impôts de ces derniers. Généralement, les chercheurs bénéficient d'un accès gratuit à ces données pour leurs recherches. Néanmoins, quand quelqu'un est extérieur à ce réseau, il lui est difficile d'obtenir ces données.

Les énergéticiens ont besoin depuis longtemps des informations météorologiques. Par exemple, ces données leur permettent de prédire la production électrique annuelle d'un panneau solaire en fonction de sa situation géographique.

Pour répondre à ce besoin de données, ils ont commencé par collecter ces informations à l'aide de stations au sol. Cette source d'information a donc commencé par être répertoriée sur des documents papiers. Par la suite, ces documents ont été reproduits et formatés sous forme de tableaux, de graphiques ou de cartes, les rendant plus faciles à analyser ou à utiliser. Les premiers à proposer ce type de documents sont ESRA [76], le WMO [72] ou le Council for Scientific and Industrial Research (CSIR) [30] ou [14, 31] pour finalement avoir des données sur un grand nombre de pays. La forme papier de ces données présentait l'avantage d'être consultable à tout moment, mais elle comporte des données de base figées. De plus, la recherche d'informations particulières y est difficile et leur utilisation demandent une fastidieuse phase de copie ou saisie de ces données. La technologie aidant, ces données se sont informatisées. WMO est le premier à proposer les données sous la forme de disquette [85]. Ces disquettes accompagnent le document, mis-à-jour pour l'occasion. Par la suite, les chercheurs ont utilisé les CD-ROM, leurs fournissant un plus grand nombre de données et des applications. Le CD-ROM de WMO [108] ou celui de Meteonorm [68, 84] en sont des exemples. Les applications fournies permettent d'obtenir des données dérivées des données de base contenues sur le CD-ROM. Ces données peuvent être présentées de la même façon que dans les publications "papier" traditionnelles, c'est-à-dire sous forme de tableau de valeurs, de graphique ou de carte. Plus tard, des sites Web spécialisés ont été créés pour accroître la diffusion des données, comme le site SoDa [95], le site Satel-Light [27] ou le site du Council for Scientific and Industrial Research [26], ce dernier proposant des données de la base SARERD (South Africa Renewable Energy Resource Database). Ces sites proposent des données pouvant être gratuites ou payantes. Les avantages de ces sites sont qu'ils offrent un accès facilité aux données, une mise à jour régulière et des frais de recopie presque nuls. Ces sites fournissent l'accès à certaines données dans le rayonnement solaire à l'aide d'interfaces internet interactives.

3.2.1 Accessibilité des données météorologiques

Accèder à l'information pertinente n'est pas facile dans de nombreux cas et pour diverses raisons [24, 32, 67, 105].

Dispersion géographique des sources de données Les données sont habituellement fournies par les bureaux météorologiques de chaque pays. Les adresses de ces différents bureaux peuvent être obtenues sur le site Web du WMO [107]. Parfois, un accès centralisé aux données météorologiques est possible. Par exemple, le centre mondial de données de radiation (WRDC) propose, pour de nombreux pays, un accès en ligne à des données archivées.

Discordance temporelle et spatiale des sources de données Les données météorologiques nécessaires ne sont généralement pas disponibles aux mêmes instants et mêmes lieux. Par exemple, les données de rayonnement diffus ne sont pas en coïncidence avec les données de rayonnement global. En disposant de ces deux données dans un même lieu, il est possible d'en déduire la quantité de rayonnement frappant une surface plane inclinée, avec une bonne précision. Dans le cas contraire, autrement dit, dans la plupart des cas, les utilisateurs doivent utiliser les données de rayonnement global et faire appel à un modèle de faible précision qui déduit les composantes directes et diffuses du rayonnement à partir des mesures du rayonnement global.

Incohérence des intervalles d'intégration Les mesures de rayonnement sont souvent faites sur une base journalière et non pas sur une base horaire, sauf pour un nombre limité de stations. Les utilisateurs doivent donc trouver des modèles qui génèrent des éclairements heure par heure à partir d'éclairements journaliers. Ces modèles font appel à des connaissances statistiques sur le profil journalier. Ces profils ne sont pas disponibles pour la plupart des stations de mesure. Ils doivent donc être déduits d'autres stations réalisant des mesures sur la base horaire.

Divergence de la forme des données L'accès aux données reste compliqué du fait de la variété des types de données : observations journalières ou observations horaires, rayonnements globaux ou diffus, durées d'ensoleillement ; ces types de données étant différents pour chaque pays. Ainsi, si ces données sont collectées par différents pays, ces données auront des disparités dans leur standard de stockage et dans leur unité. Par exemple, certains pays préfèrent utiliser des Jm^{-2} quand d'autres vont utiliser Jcm^{-2} ou Whm^{-2}. Le département de l'énergie des Etats Unis d'Amérique (DoE) a défini des unités qui intègrent la définition de la production, ce qui ajoute à la confusion. Par exemple, DoE exprime les moyennes mensuelles de rayonnement sur un jour en $Wh/m^2/jour$; l'unité "jour" est ajoutée pour exprimer le fait que cette donnée est une énergie pour une journée. Mais l'interprétation de la physique conventionnelle de telles unités fait de cette quantité une densité de puissance dont l'unité correcte est W/m^2. Ces unités du DoE sont abondamment utilisées aux Etats Unis d'Amérique par les serveurs Web situés dans ce pays. Par exemple, la NASA les utilise pour le service Surface Solar Energy Data Set.

Il existe plusieurs façons d'exprimer le temps : temps universel, temps solaire moyen, temps solaire réel, temps local. Un standard existe pour la définition des données horaires. Ce standard est défini par WMO : l'heure attribuée à une donnée

correspond à la fin de la période de mesures, c'est-à-dire, qu'une donnée horaire mesurée à 12 h a été réalisée entre 11 h et 12 h. Ceci n'est pas le cas dans d'autres réseaux, où le temps associé à la mesure peut représenter le début de la période ou le milieu.

La diversité s'accroît avec la disponibilité de données sous la forme de grille, comme les données dérivées des mesures satellitaires ou les sorties de modèles numériques. La taille des surfaces représentées par l'information peut changer selon les capteurs des satellites ou la taille des mailles de calcul. La fréquence des observations est variable en fonction de la source de l'information ainsi qu'en fonction de la période d'observation.

Enfin, les données sont payantes en fonction de la situation des pays. Par exemple, les données météorologiques peuvent être achetées à des prix très bas, elles peuvent même parfois être gratuites. En Europe et dans certain pays asiatiques, le coût est associé à l'information. Ce prix est généralement peu élevé, mais peut devenir important pour des requêtes qui demandent de longues séries temporelles sur un grand nombre de stations météorologiques. Les données peuvent être obtenues dans chacun des différents bureaux responsables de la zone cherchée.

Pour conclure cette section, il y a un important besoin d'offrir un accès harmonisé aux informations météorologiques. Le Web est actuellement le meilleur candidat pour répondre à ce besoin. L'utilisateur devrait avoir accès à un unique interlocuteur possédant un système permettant de rassembler les informations nécessaires pour utiliser de multiples sources de données réparties géographiquement.

3.2.2 Les services Web météorologiques actuels

Comme le montre l'historique un grand nombre de données sont disponibles plus ou moins facilement. Parmi ces données, une petite quantité l'est sous la forme de services Web. Je présente dans cette section les services Web disponibles en météorologie et reconnus des professionnels ; cette liste n'est certainement pas exhaustive compte-tenu que des services Web sont ajoutés et supprimés régulièrement. Je distingue quatre catégories de services Web :

- les services Web d'observation ;
- les services Web fournissant des données dérivées ;
- les services Web de conversion d'unités ;
- les services Web de données climatiques.

Les services Web d'observation du rayonnement solaire sont actuellement au nombre de quatre. Ils fournissent tous des séries temporelles d'éclairement global total sur plan horizontal. Le service Web Solemi appartient au Deutsches LuftRaumfahrt (DLR). Il fournit des séries au pas horaire en 2005 sur l'Europe. Le service Web Helioclim-1 fournit des séries au pas journalier entre 1985 et 2005 sur la zone Europe et Afrique. Le service Web Helioclim-3 fournit des séries au pas de 15 min à partir de 2004 sur la zone Europe et Afrique. Ces deux derniers services Web sont assurés par le CEP de Mines ParisTech. La NASA propose un service Web, nommé SSE, qui fournit des séries au pas journalier de 1983 à 2003 qui couvre le monde entier. La résolution spatiale des données fournies est définie par la surface au sol que représente une mesure. Cette résolution est variable : de 100×100 km^2 pour SSE à 5×5 km^2 pour Solemi et Helioclim-3. L'accès aux données de Solemi, Helioclim-1 et Helioclim-3 est gratuit pour certaines années, et payant pour les autres. L'accès à SSE est entièrement gratuit. Pour tous ces services Web, les entrées sont : une période et une position géographique.

Le service Web NCEPTemperature, créé par Mines ParisTech, exploite les réanalyses météorologiques effectuées par le NCEP (National Centers for Environmental Prediction) aux Etats-Unis d'Amérique. Ces données sont disponibles gratuitement ; elles couvrent le monde et débutent en 1945. Les entrées de ce service sont la position géographique et une période. La sortie de ce service Web est une série temporelle de températures moyennes au pas journalier. La résolution spatiale est de $180 \times 180\ km^2$.

Les services Web fournissant des données dérivées sont au nombre de deux. Le premier est un service Web, nommé IrradInclinedPlane, qui calcule des séries temporelles d'éclairement sur un plan incliné. Le second, IrradNormalPlane, calcule des séries temporelles d'éclairement sur un plan normal aux rayons du soleil. Ces services Web ont pour entrées une position géographique et une série temporelle d'éclairement global total sur plan horizontal. Le premier a des entrées supplémentaires : l'orientation et l'inclinaison du plan. Ces deux services Web n'ont pas de limite géographique, ni temporelle. Ces deux services Web appartiennent au CEP de Mines ParisTech.

Les services Web convertisseurs d'unités sont utilisés dans différents domaines. On peut en trouver sur le site WebServiceX [112], par exemple. Ces services Web n'ont pas de limitation sauf celle inhérente à l'unité utilisée, par exemple une valeur en degré Kelvin n'a pas de sens en dessous de zéro. Habituellement, ces services Web prennent une valeur d'entrée et fournissent le résultat converti. Certains d'entre eux peuvent avoir pour entrée une liste de valeurs, et donc convertir toutes les valeurs de la liste. D'autres sont capables de faire plusieurs conversions différentes et donc utilisent un paramètre d'entrée indiquant quelle conversion effectuer. L'un des services Web les plus utilisés de SoDa est le service Web sexa2decimal qui convertit des degrés sexagésimaux en degrés décimaux.

Le service Web EMPClimate, de Mines ParisTech, fournit des données relatives au climat sous forme de moyennes mensuelles intégrées de 1985 à 2004 en tout point du globe. Outre l'éclairement, les paramètres météorologiques fournis sont la température et l'humidité relative et précisément les minima, maxima et moyennes. L'entrée de ce service est la position géographique.

3.3 Besoins en météorologie

L'expérience des premiers usages des services web en météorologie ont montré certaines limites des services web et des souhaits d'amélioration ont été exprimés. Les objets manipulés en météorologie, comme par exemple, la température de l'air, sont des objets dynamiques, offrant des variations importantes dans l'espace et dans le temps. Leur représentation est souvent incomplète ou fragmentaire. Par exemple, des mesures en un point peuvent être interrompues durant des périodes d'étalonnage des instruments, des résultats de modèles numériques complexes sont disponibles sur une grille à maille large pour telle région d'un continent et sur une maille plus fine pour telle autre région. Disposer de séries de mesures complètes ou de résultats de modèle couvrant un continent avec la maille la plus fine possible est un exemple d'amélioration désiré régulièrement par les météorologues.

La composition automatique et adaptative apporte des réponses aux souhaits d'amélioration de la composition. Ces améliorations se scindent en deux types : des améliorations générales à la composition des services Web décrites dans les trois premières sections, et des améliorations spécifiques à la météorologie mais pouvant également s'appliquer à d'autres domaines des géosciences décrites dans les trois dernières

sections.

3.3.1 Validité et justesse de la composition de services Web

Il est nécessaire de distinguer des services Web intrinsèquement différents entre eux afin de pouvoir qualifier la validité d'un résultat selon la fonctionnalité ou l'opération demandée par l'utilisateur.

Une première qualification de la validité d'un résultat est tout simplement la vérification de la concordance des types de retour des opérations. Par exemple, deux services donnant des séries temporelles, l'un de la température de l'air et l'autre de rayonnement solaire, devraient bien évidemment être différenciés parce que les quantités physiques qu'ils manipulent et retournent sont de nature différente, même si le type informatique du résultat est le même.

Pour pouvoir effectuer une telle vérification, il est nécessaire de disposer d'une liste des types de données. Des efforts importants sont actuellement réalisés pour la définition uniformisée et standardisée de tous les types de données intervenant dans le domaine de la météorologie et plus généralement des sciences de l'environnement et de la géographie. Cette action, menée à l'échelle internationale, s'illustre par des initiatives prises par le G8, comme le GEOSS (Global Earth Observation System of Systems) [37], ou par l'Agence Internationale de l'Energie [48], ou encore des projets européens comme MESoR [67] et plus anciennement SoDa [95].

Dans ce cadre de standardisation, une activité importante est l'établissement d'un thésaurus, qui regroupe l'ensemble des types de données utilisées en météorologie. Il existe un thésaurus réalisé par l'Open Geospatial Consortium (OGC) [70] qui traite un très large éventail de types de données de manière très générique. Plusieurs problèmes sont apparus lors de son utilisation pratique par un groupe de chercheurs européens dans le cadre de la météorologie pour l'énergie solaire. D'une part, l'expression générique des objets éloigne ces objets conceptuels de la perception qu'en ont les praticiens et la complexité résultante de cette conceptualisation rend ce thésaurus difficilement utilisable par les météorologues. D'autre part, il y a très peu d'outils de gestion des services Web implantant totalement ce thésaurus et des problèmes techniques liés à l'utilisation de tels outils l'ont rendu inutilisable en l'état pour mon travail.

Par conséquent, l'un de mes premiers travaux de thèse a consisté en la recherche d'autres thésaurus. Il en existe plusieurs développés par des organisations météorologiques, comme l'Organisation Mondiale de la Météorologie, le bureau météorologique australien, ou l'agence en charge de la météorologie (NOAA) aux Etats-Unis [42]. Le premier s'appuie sur le thésaurus de l'OGC et n'a pas encore d'existence pratique ; les autres sont très liés aux moyens d'observation et à la collecte des données dans les stations météorologiques. J'ai donc collecté les différents types de données auprès du groupe de chercheurs européens, cité plus haut, et j'ai instauré une terminologie commune afin de les regrouper au sein d'un thésaurus approprié à la météorologie pour l'énergie solaire [42].

Outre la concordance des types de retour, une autre qualification possible de la validité consiste en la différenciation des sémantiques des opérations. Par exemple, une opération retournant la température des océans doit, suivant les applications, revêtir un sens différent d'une opération qui retournerait la température de l'air ou du soleil.

Par conséquent, la composition des fonctionnalités offertes pour la météorologie doit prendre en compte de manière précise autant les différents types des données

manipulées que la sémantique des opérations les manipulant.

3.3.2 Reconfiguration en cas de panne

Comme je le décris dans la section 2.1, les services Web sont, le plus souvent, des ressources hors de contrôle de leurs utilisateurs. Ces ressources peuvent parfois devenir indisponibles, pour diverses raisons comme une panne matérielle, la congestion du réseau ou encore la suppression ou la restriction d'accès de la ressource par son fournisseur.

Les météorologues souhaitent minimiser l'impact de la disparition des ressources, et donc en cas de défaillance d'un service Web, ils désirent remplacer le service manquant par un ou une combinaison de plusieurs services Web. Pour cette opération, il est naturellement nécessaire de s'assurer que le comportement obtenu est *équivalent* [29] au comportement initial requis.

Prenons des exemples concrets qui illustrent deux facettes du besoin de substitution. Un service Web fournissant une série de températures en degrés Celsius, peut être remplacé par la composition d'un service Web fournissant une série de températures en Fahrenheit et d'un service Web assurant la conversion d'unité, de Fahrenheit en Celsius. Dans un autre exemple, on peut remplacer un service Web offrant des données à basse résolution spatiale par une composition d'un service Web offrant des données à haute résolution spatiale et d'un service Web effectuant des opérations de moyenne spatiale.

3.3.3 Prise en compte des nouveaux services Web

Comme nous l'avons vu dans la section 2.1, Internet est en perpétuelle évolution. Les fournisseurs proposent continuellement de nouveaux services Web qui peuvent être pris en compte pour, par exemple, pallier les défaillances des services Web, améliorer la qualité des données, proposer de nouvelles fonctionnalités grâce aux nouvelles compositions possibles.

Il faut donc définir un catalogue de services Web appliqué à la météorologie mis à jour régulièrement et de manière automatique pour que mes méthodes de composition puissent prendre en compte les nouvelles applications décrites sous la forme de services Web. Ce catalogue doit comporter les données nécessaires pour les opérations de composition, par exemple les types des entrées et des sorties correspondant au thésaurus ou encore la sémantique des opérations. La forme de ce catalogue varie naturellement en fonction des méthodes de composition.

A terme, l'idée est de proposer un catalogue et une mise à jour automatique par les fournisseurs de services Web en météorologie, à la manière des annuaires de services UDDI [101] qui référencent des services Web à l'aide de descriptions en XML afin d'en permettre la localisation et leur utilisation sur le réseau. Dans le cadre de ma thèse, je me suis restreint à la réalisation d'un catalogue statique que je mets à jour manuellement au fur et à mesure de mes avancées dans les techniques de composition.

3.3.4 Complétion des données

La complétion des données consiste à ajouter aux données en sortie d'un service Web, les données manquantes fournies par un autre service Web. L'idéal est d'obtenir la même qualité de données mais une complétion peut dégrader la qualité des données.

Par exemple, lorsqu'un service Web fournit une série temporelle comportant des trous, ou manques, cette série temporelle pourra éventuellement être complétée en utilisant d'autres services Web à la couverture différente, ou à une qualité de données moindre, ce qui explique qu'ils n'ont pas été choisis comme service initial.

La traçabilité des données doit être assurée.

3.3.5 Amélioration de la qualité

Comme dans d'autres domaines, fournir des données n'est pas suffisant si elles ne sont pas fournies avec des paramètres de qualité, comme des marges d'erreurs. Fournir les paramètres de qualité est une condition pour pouvoir utiliser le service Web de façon sure. L'amélioration de la qualité consiste à prendre en compte différents critères de qualité des données, fournis par les services Web, tels que des intervalles de confiance sur une série d'observations, ou le taux de données manquantes, ou encore le coût financier de la mise à disposition des données.

Plusieurs auteurs ont souligné l'importance de la qualité de service [47, 51, 60]. On pourrait ainsi soit optimiser le choix des services suivant ces critères de qualité parmi un ensemble de services équivalents, soit sélectionner des services qui viendraient améliorer a posteriori la qualité des sorties d'un autre service.

La validation de la qualité des données est un problème ouvert en météorologie et le restera pour longtemps, principalement parce que l'on manque de références claires dans la plupart des cas [24]. La composition de services Web ne simplifie pas cette tâche car, dans ce cas, la propagation de la qualité doit être prise en compte de façon dynamique.

3.3.6 Estimation de la qualité de la sortie finale

L'estimation de la qualité de la sortie finale consiste à combiner les attributs de qualité des différents services Web composés afin d'obtenir cette estimation.

Cette estimation peut, par exemple, s'exprimer sous la forme d'un écart-type au résultat final pour définir une incertitude, ou encore s'exprimer en une valeur qui représente un coût financier.

Cet attribut qualitatif peut également servir au choix de la meilleure composition vis-à-vis de la requête.

Dans le cas de la météorologie, composer des services Web tout en tenant compte de la qualité des données échangées est une tâche complexe. Il faut tout d'abord définir de manière précise les différents types de qualité devant être pris en compte. Il faut ensuite s'assurer de la cohérence de leur composition.

Ce travail, en dehors de la portée de cette thèse, n'y sera pas abordé mais pourra constituer une extension intéressante de ces travaux.

3.4 Propriétés des services Web météorologiques

Les services Web météorologiques sont un sous-ensemble des services Web bien adaptés à la composition des services Web comme nous allons le voir dans la suite de cette section qui décrit les caractéristiques des services Web et la manière de les formaliser.

Nous déterminerons tout d'abord le mode d'appel des opérations offertes au travers des services Web en météorologie et instruirons les propriétés de ces opérations. Nous les assimilerons à des fonctions mathématiques les représentant.

3.4.1 Mode de fonctionnement des opérations de services Web en météorologie

Les services Web sont accessibles au travers d'appels d'opérations.

Le WSDL définit quatre modes de fonctionnement pour ces opérations : *notification*, *one-way*, *solicit-response* et *request-response*.

Une opération de *notification* envoie un message et n'attend pas de message en retour.

Une opération en mode *one-way* reçoit un message en entrée et ne renvoie pas de réponse.

Une opération en mode *request-response* attend un message avant de renvoyer un message en réponse.

Une opération en mode *solicit-response* commence par envoyer un message et attend le message de réponse correspondant en retour.

Les opérations en météorologie fonctionnent intrinsèquement sur un mode *request-response* puisqu'une requête va toujours consister à envoyer des données d'entrée, pour y effectuer des calculs et en récupérer le résultat.

J'aborde ainsi le problème de la composition d'opérations s'effectuant en mode request-response. Dans ce mode de fonctionnement, il existe deux manières d'appeler les opérations de services Web. La première manière est de type asynchrone c'est-à-dire : plusieurs services Web peuvent être lancés au même moment et s'exécuter de manière indépendante tout en continuant l'exécution de la requête principale. La seconde est de type synchrone, c'est-à-dire qu'on attend la réponse d'une requête avant de lancer une nouvelle requête ou de poursuivre l'exécution de la requête principale.

Afin d'éviter de traiter ici les problèmes complexes liés à l'exécution asynchrone des services Web et à leur resynchronisation lors de l'exploitation des résultats, j'ai choisi de me concentrer sur la composition synchrone et séquentielle des services Web. En effet, mon travail porte sur la composition de services Web. L'approche asynchrone n'est pas nécessaire à ce travail, sachant que tout programme asynchrone sur une mémoire finie peut être traduit en un programme synchrone équivalent. Un programme synchrone peut être vu comme un programme asynchrone pour lequel les opérations asynchrones sont instantanées. Cette approche pourra être étudiée ultérieurement à mon travail de thèse.

3.4.2 Les types d'opérations réalisées par ces services

Il existe deux grands types d'opérations possibles : les unes sont des algorithmes et les autres sont des opérations fournissant de nouvelles données.

Ces deux types d'opérations sont dits sans-état (state-less). Un service Web sans état est un service qui ne mémorise pas d'information de session. A l'opposé, les services Web avec-état (state-full) enregistrent et utilisent ces informations, pour, par exemple, retenir un nom d'utilisateur, un mot de passe, des informations sur les préférences de l'utilisateur ou encore des données de retour des opérations de services Web de météorologie.

Les services Web en météorologie ont pour but :

- d'accéder à des données ;
- de ne pas permettre de modifier les données qu'ils utilisent.

Par exemple, ces services ne permettent pas d'ajouter des données dans une base de données. Par conséquent, il n'est pas nécessaire de connaître l'état ou le contexte de chaque service Web pour prévoir leurs résultats. Le message de sortie de chaque opération d'un service Web dépend ainsi directement du message d'entrée.

Naturellement, les météorologues procèdent régulièrement à la mise à jour des données qu'ils possèdent et proposent à des utilisateurs distants. Ces mises à jour seront traitées comme les mises à jour de services Web, c'est à dire, comme la disparition d'un service Web, resp. d'une base de données, et la création d'un nouveau service Web mis à jour, resp. d'une base de données mise à jour. Par conséquent, un service Web est considéré comme déterministe. De même, une base de données est considérée comme constante. Autrement dit, le service Web appelé avec les mêmes données d'entrée fournit toujours les mêmes données de sortie. La propriété de déterminisme nous permet de prévoir le résultat des appels des services Web et donc de réaliser la composition.

3.4.3 Un service Web comme une fonction partielle

Les particularités des opérations des services Web en météorologie présentées me permettent de les assimiler à des fonctions partielles pour un instant donné.

Une fonction partielle qui part de l'ensemble $\mathbf{X}$ vers l'ensemble $\mathbf{Y}$ est une relation $f \subseteq \mathbf{X} \times \mathbf{Y}$ telle que :

$$\forall x, y, y'.((x, y) \in f \ \& \ (x, y') \in f \Rightarrow (y = y')) \tag{3.1}$$

Dans le cas des services Web, leurs opérations manipulent des entrées et des sorties que j'appellerai *messages*. Chaque opération en météorologie pour être exécutée nécessite donc un message qui contient les informations nécessaires à son exécution. Ce message, une fois transmis à l'opération, est traité et l'opération renvoie un autre message contenant les données de la réponse.

Je définis donc les deux ensembles suivants :

- l'ensemble $\mathbf{M}$ des messages ;
- l'ensemble $\mathbf{S}$ des opérations de services Web avec $\mathbf{S} \in \mathcal{P}(\mathbf{M} \times \mathbf{M})$.

Je pose ici la terminologie qui sera utilisée dans la section suivante concernant la formalisation de la composition de services Web pour la météorologie.

3.5 Formalisation de la composition de services Web en météorologie

Cette section présente la formalisation mathématique de la composition des services Web en météorologie.

Dans une première partie je discute rapidement de l'intérêt de la sémantique des langages de programmation et introduit succinctement la sémantique dite opérationnelle. Dans une seconde partie je présente en détail la formalisation, sous la forme d'une sémantique opérationnelle, du langage de composition de service Web. Cette sémantique sera proposée dans le cas d'un environnement idéal où les services existent et sont disponibles puis étendue aux cas d'environnements réels où les services

peuvent être indisponibles, avoir une couverture partielle, ou renvoyer des erreurs. Enfin j'étends cette sémantique à la prise en compte de la qualité de service.

Dans une dernière partie je discute des objectifs de ma thèse pour les prochains chapitres de cette thèse.

3.5.1 Introduction à la sémantique opérationnelle

La sémantique des langages de programmation est une approche qui permet de donner une signification mathématique rigoureuse aux programmes informatiques. Elle définit la façon dont les programmes écrits dans un langage donné doivent être interprétés et permet donc de lever toute ambiguïté dans l'interprétation du comportement des programmes.

Il existe plusieurs manières de donner un sens formel à un langage de programmation : la sémantique opérationnelle, la sémantique dénotationnelle et la sémantique axiomatique.

La sémantique dénotationnelle associe une fonction mathématique (appelée dénotation) à chaque programme, qui représente la signification du programme.

Dans la sémantique axiomatique, le programme est considéré comme transformant des propriétés logiques sur l'état de la mémoire.

Enfin, dans la sémantique opérationnelle, la signification d'un programme est la suite des états de la machine qui exécute le programme. C'est cette sémantique que nous utiliserons pour la formalisation de notre langage de composition de services web en météorologie. Notre langage permettant d'écrire des programmes fonctionnels, l'état final d'une suite qui termine donnera donc la valeur de retour du programme. Nous verrons aussi que notre sémantique étant déterministe, il ne peut y avoir qu'une seule suite de calculs et qu'une seule valeur de retour d'un programme.

Classiquement, la sémantique opérationnelle permet de définir rigoureusement la signification d'un programme sous la forme d'un système de transition d'états qui rend compte du comportement attendu d'un programme du langage.

Cette définition permet tout autant une analyse formelle d'un programme (par exemple par des méthodes de model-checking [34]) que de comparer des programmes entre eux. Ainsi, donner une sémantique formelle à nos opérations de composition va nous permettre d'utiliser des méthodes de preuves d'équivalences de compositions.

Une sémantique opérationnelle prend la forme d'un ensemble de règles qui vont décrire les transitions valides du système. Nous présentons ici notre sémantique dans le style des règles de réécriture introduites par Plotkin [78] : à partir d'un programme P et d'une entrée I, la règle de réécriture va déterminer la sortie O correspondante et le nouveau programme P' adapté à traiter les entrées restantes.

Le comportement d'une composition sur une entrée va être calculé de manière pas à pas. Cette sémantique présente donc l'avantage d'être proche de l'implémentation de l'interpréteur du langage.

3.5.2 Sémantique opérationnelle du langage de composition de services Web

Cette section présente la formalisation dans une sémantique opérationnelle de mon langage de composition de services Web. Pour construire les bases de ce langage, je me suis naturellement inspiré de la composition de services Web actuellement réalisée par SoDa. SoDa réalise simplement l'enchaînement d'appels de services Web, consistant

à faire appel à des opérations de services Web les unes après les autres, je l'ai donc étendu pour traiter des cas plus complexes.

Je présente ici, la formalisation dans le cas idéal. Le cas idéal est le cas où tous les services Web fonctionnent normalement et sont disponibles.

3.5.2.1 Introduction du langage de composition de services Web

Pour répondre aux besoins des météorologues dans ce cas idéal, je définis trois opérateurs qui sont l'appel de services, la séquence et l'union entre deux compositions de services Web. Ces trois opérateurs sont des compositions. L'appel de service Web fournit ses entrées à un service et le lance ; c'est la composition de base. La séquence est un enchaînement de compositions de services Web consistant intuitivement à faire appel à des opérations de services Web les unes après les autres. L'union entre deux compositions de services Web doit évaluer les deux compositions et faire l'union des deux messages de sortie. L'idée de cet opérateur est de permettre la complétion des données.

3.5.2.2 Syntaxe du langage de composition de services Web

Dans cette section, je définis la syntaxe des opérateurs de mon langage de composition de services Web.

L'ensemble des opérations de service Web est répertorié dans **S**. L'ensemble des compositions de services Web peut être décrit à l'aide d'expressions à la manière des expressions mathématiques. L'ensemble des expressions **Cexp** qui représente l'ensemble des compositions que je réalise se décrit récursivement à l'aide de l'ensemble des expressions de compositions **Cexp** et de l'ensemble **S** des opérations de service Web

3.5.2.3 Règles de construction de la syntaxe

Les règles de construction de l'ensemble des compositions de services Web peuvent être représentées par une variante du BNF (Backus-Naur Form) où le symbole := se lit *peut être* et le symbole | se lit *ou*.

Dans la suite de cette section, s représente une opération d'un service Web et c_i une expression de composition de **Cexp**.

3.5.2.4 Syntaxe de l'appel d'opération composition

$$c := s \tag{3.2}$$

Cette règle représente *la composition particulière qui consiste à faire appel à une opération d'un service Web représenté par la fonction s* cela définit que toute opération de services Web est une composition.

3.5.2.5 Syntaxe de la séquence

Je définis l'évaluation de la séquence de compositions, représentée par l'expression $c_0 \circ c_1$. Je peux expliquer intuitivement cette évaluation avec les étapes suivantes :

1. évaluer la composition c_0 dans l'environnement Γ avec le message m pour obtenir le message de retour m' ;

2. puis évaluer la composition c_1 dans l'environnement Γ avec le message m' pour obtenir le message m'' ;
3. enfin retourner le message m''.

$$c := c_0 \circ c_1 \tag{3.3}$$

Cette règle exprime que *si* c_0 *et* c_1 *sont des expressions de composition alors la séquence entre* c_0 *et* c_1 *exprimée* $c_0 \circ c_1$ *est une expression de composition.* $c_0 \circ c_1$ est la séquence au sens mathématique des deux compositions c_0 et c_1, la composition c_1 est évaluée, elle retourne un message qui sera utilisé pour la composition c_0.

3.5.2.6 Syntaxe de l'union

Je définis l'évaluation de l'union de compositions, représentée par l'expression $c_0 \uplus c_1$. Je peux expliquer intuitivement cette évaluation avec les étapes suivantes :

1. évaluer la composition c_0 dans l'environnement Γ avec le message m pour obtenir le message de retour m' ;
2. puis évaluer la composition c_1 dans l'environnement Γ avec le message m' pour obtenir le message m'' ;
3. enfin fusionner les messages m' et m'' et renvoyer le résultat.

3.5.2.7 Syntaxe du langage complet

La règle de construction des expressions de composition s'écrit de la façon suivante :

$$c := c_0 \circ c_1 \mid s \mid c_0 \uplus c_1 \tag{3.4}$$

où c_0, c_1 et c appartiennent à **Cexp** et s appartient **S**.

3.5.2.8 Sémantique opérationnelle du langage de composition de services Web

Dans la section précédente, j'ai défini les constructions syntaxiques de mon langage. Dans cette section, je vais définir la signification de chacune de ces constructions syntaxiques en utilisant, pour cela, la sémantique opérationnelle. Je vais donc utiliser ce formalisme pour définir le comportement des compositions de services Web souhaitées par les météorologues et ainsi soulever les difficultés que pose la réalisation de la composition automatique et adaptative de services Web pour la météorologie.

Les règles et leur environnement Considérons l'évaluation d'une expression de composition c sur le message d'entrée m dans l'environnement Γ. Je peux représenter la situation de la composition c en attente d'exécution sur le message m par le triple (Γ, m, c) que je note $\Gamma, m \vdash c$. Je peux donc définir une relation d'exécution entre cette paire et le message m résultant :

$$\Gamma, m \vdash c \rightarrow r \tag{3.5}$$

Ce qui signifie que la composition c dans l'environnement Γ avec le message m en entrée retourne le message r. L'environnement Γ comporte les données omniscientes

nécessaires à l'évaluation de la composition. Les besoins en météorologie me permettent de définir un environnement constant tout au long de l'exécution. L'environnement comporte alors seulement l'ensemble des opérations de services Web S connus.

Bien que cette description soit informelle, elle décrit comment évaluer la composition à partir de l'évaluation des sous compositions. Cette description peut être traduite formellement en :

$$\frac{\Gamma, m \vdash c_1 \rightarrow m' \quad \Gamma, m' \vdash c_0 \rightarrow m''}{\Gamma, m \vdash c_0 \circ c_1 \rightarrow m''} \tag{3.6}$$

et se lit : Si $\Gamma, m \vdash c_1 \rightarrow m'$ et $\Gamma, m' \vdash c_0 \rightarrow m''$ alors $\Gamma \vdash c_0 \circ c_1 \rightarrow m''$.

Les règles ont une prémisse et une conclusion et j'ai suivi la pratique habituelle en écrivant la règle avec la prémisse au-dessus de la barre horizontale et la conclusion en dessous.

Les règles sont utilisées lors des dérivations, où les faits en dessous de la barre horizontale sont dérivés des faits au-dessus.

Certaines règles n'ont pas besoin de prémisse et sont appelées axiomes.

Sémantique des appels aux opérations des services Web Dans ma sémantique c'est un axiome qui définit l'appel d'un service Web :

$$\frac{}{\Gamma, m \vdash s \rightarrow m'} \text{avec } (m, m') \in s \tag{3.7}$$

Cet opérateur consiste à appeler le service Web s avec le message m. Le résultat de cet appel produit le message m'.

Sémantique de l'opérateur de séquence $\circ$ L'opérateur $\circ$ consiste, tout comme celui utilisé par SoDa à réaliser l'enchaînement des opérations de services Web, ce que définit la règle suivante :

$$\frac{\Gamma, m \vdash c_1 \rightarrow m' \quad \Gamma, m' \vdash c_0 \rightarrow m''}{\Gamma, m \vdash c_0 \circ c_1 \rightarrow m''} \tag{3.8}$$

Sémantique de l'opérateur d'union $\uplus$ La sémantique de l'opérateur d'union est la suivante :

$$\frac{\Gamma, m \vdash c_0 \rightarrow m' \quad \Gamma, m \vdash c_1 \rightarrow m''}{\Gamma, m \vdash c_0 \uplus c_1 \rightarrow \mathcal{U}(m', m'')} \tag{3.9}$$

où $\mathcal{U}$ est une fonction chargée d'unir deux messages m et m'. Cette union est une spécificité des besoins en météorologie qui manipule, par exemple, des séries temporelles. La réalisation de cette fonction, pour qu'elle soit intéressante, n'est pas triviale.

Une première version de cette fonction peut consister simplement à systématiquement choisir un des deux messages d'entrée, autrement dit :

$$\forall m_0, m_1 . (m_0, m_1) \in M \times M \ \& \ \mathcal{U}(m_0, m_1) = m_0 \tag{3.10}$$

Comme nous le voyons cette fonction n'est pas satisfaisante et revient à annuler l'opération d'union pour la remplacer par m_0. Par la suite je montrerai comment améliorer cette fonction.

3.5.3 Extension de la composition des services Web au cas réel

Nous allons dans les premières sections décrire les aléas de la composition des services Web dans un environnement réel.

3.5.3.1 Non-déterminisme de l'environnement

Internet et les services Web ne sont pas contrôlés par leurs utilisateurs, par conséquent du point de vue des utilisateurs Internet et les services Web ont parfois des comportements aléatoires.

Par exemple, un fournisseur de services Web peut choisir de supprimer un service Web ou de le modifier, de même pour le réseau Internet, ce dernier peut subir des modifications entraînant des perturbations dans le transfert de données pouvant jusqu'à rendre inaccessibles des services Web qui l'étaient jusqu'à présent. Cet aléa a été constaté lors de l'utilisation de l'application SoDa par les météorologues, et ils ont donc exprimé le besoin de pouvoir remplacer les services Web par d'autres. Par ailleurs, cet aléa justifie l'intérêt de la composition automatique et adaptative. Pour prendre en compte cet aléa il faut d'abord modifier la représentation des opérations de services Web.

3.5.3.2 Domaine de définition partiel des services Web

De plus, dans la pratique, le domaine de définition des services Web est partiel vis-à-vis de l'ensemble des messages. Il faut donc prendre en compte le fait que les messages d'entrée peuvent être hors du domaine de définition du service Web.

3.5.3.3 Traitement des aléas

En définitive, ces aléas peuvent être considérés comme des erreurs et deux types d'erreurs peuvent survenir : les erreurs provoquées par le services Web lui même et les erreurs consécutives à l'appel du service Web. Les erreurs des services Web surviennent généralement lorsque les entrées fournies à l'opération du service Web ne sont pas valides, par exemple le type des données fournies est inconnu, ou l'opération du service Web est indéfinie pour ce message d'entrée. Les autres erreurs sont provoquées par l'état d'Internet. Par exemple, lorsque le réseau Internet est encombré à l'un de ces noeuds, les messages passant par ces noeuds peuvent arriver après le temps de réponse maximum alloué. Je rassemble ces messages d'erreurs dans l'ensemble **Error**. Je définis donc les appels aux différentes opérations des services Web comme des fonctions totales qui partent de $\mathbf{M}$ et vont vers l'ensemble $\mathbf{M} \cup \mathbf{Error}$. Une fonction totale $f \subseteq \mathbf{X} \times \mathbf{Y}$, que j'appellerai fonction par la suite, est une fonction partielle pour qui : pour tous les éléments de l'ensemble de départ $x \in \mathbf{X}$ il existe $y \in \mathbf{Y}$ tel que $(x, y) \in f$. Cette représentation n'est valable que pour un instant donné. Par la suite de cette section, je considère les compositions à un instant donné, et donc un même service Web à cet instant est déterminé par un ensemble $s \in \mathcal{P}(\mathbf{M} \times \{\mathbf{M} \cup \mathbf{Error}\})$

3.5.4 Extension de la sémantique à la prise en compte des aléas

La sémantique dans le cas idéal ne prend pas en compte les messages d'erreur qui peuvent survenir, pour cette raison je dois la compléter. Je commence par introduire un nouvel opérateur de choix exclusif, que je note $\oplus$, et qui répond au besoin de

pouvoir remplacer des compositions défaillantes par d'autres. Cet opérateur binaire prend deux compositions en paramètre et réalise la première composition, si cette dernière renvoie une erreur alors la seconde est exécutée, dans le cas contraire la seconde est ignorée. Les compositions sont donc construites de la façon suivante :

$$c := c_0 \circ c_1 \mid s \mid c_0 \uplus c_1 \mid c_0 \oplus c_1 \tag{3.11}$$

Sémantique du choix exclusif Dans le cas idéal la sémantique du choix exclusif ignore toujours la seconde composition. Elle est donc définie par :

$$\frac{\Gamma, m \vdash c_0 \to m'}{\Gamma, m \vdash c_0 \oplus c_1 \to m'} \tag{3.12}$$

Les messages d'erreur et l'opérateur de choix exclusif En suivant le comportement décrit plus haut, je définis deux règles complémentaires :

$$\frac{\Gamma, m \vdash c_0 \to err \quad \Gamma, m \vdash c_1 \to m''}{\Gamma, m \vdash c_0 \oplus c_1 \to m''} \tag{3.13}$$

$$\frac{\Gamma, m \vdash c_0 \to err \quad \Gamma, m \vdash c_1 \to err'}{\Gamma, m \vdash c_0 \oplus c_1 \to err'} \tag{3.14}$$

Les messages d'erreurs et l'opérateur d'appel de service Web Le cas de l'appel à un service Web n'est pas très différent de l'appel des services Web dans le cas idéal, compte tenu qu'ils sont vus maintenant comme des fonctions totales, on peut juste ajouter la règle suivante :

$$\frac{}{\Gamma, m \vdash f \to err}(m, err) \in f \tag{3.15}$$

Prise en compte des erreurs de l'opérateur d'union L'opérateur d'union se charge simplement de propager les erreurs :

$$\frac{\Gamma, m \vdash c_0 \to m' \quad \Gamma, m \vdash c_1 \to err}{\Gamma, m \vdash c_0 \uplus c_1 \to err} \tag{3.16}$$

$$\frac{\Gamma, m \vdash c_0 \to err \quad \Gamma, m \vdash c_1 \to m'}{\Gamma, m \vdash c_0 \uplus c_1 \to err} \tag{3.17}$$

$$\frac{\Gamma, m \vdash c_0 \to err \quad \Gamma, m \vdash c_1 \to err'}{\Gamma, m \vdash c_0 \uplus c_1 \to err'} \tag{3.18}$$

Prise en compte des erreurs pour la séquence de services Web Tous comme pour l'union, les erreurs qui surviennent sont propagées :

$$\frac{\Gamma, m \vdash c_1 \to err}{\Gamma, m \vdash c_0 \circ c_1 \to err} \tag{3.19}$$

$$\frac{\Gamma, m \vdash c_1 \to m' \quad \Gamma, m' \vdash c_0 \to err}{\Gamma, m \vdash c_0 \circ c_1 \to err} \tag{3.20}$$

Limite de cette modélisation Cette modélisation considère les erreurs comme déterministes, hors par nature les erreurs des services Web peuvent être aléatoires, en particulier les erreurs où le service est en panne, ou ne répond pas. Ce type d'erreur dépendant de l'état du réseau Internet. Cet aléa se répercute dans l'évaluation des compositions, et justifie la forme du choix exclusif.

3.5.5 Prise en compte de la qualité dans la formalisation

Les météorologues souhaitent prendre en compte la qualité des données, ainsi que celle des services Web pour choisir de façon intelligente les compositions et les services Web les plus appropriés. Plusieurs axes se dégagent de cette prise en compte. Le premier axe se porte sur la granularité de la qualité, faut-il définir la qualité pour un service Web donné, une opération donnée, un message donné ou une simple valeur donnée. Le second axe se porte sur la requête, faut-il laisser l'utilisateur choisir un critère de qualité, ou faut-il que ce critère soit pré-défini. Les réponses à ces questions peuvent changer la méthode de composition, pour cette raison je vais d'abord m'intéresser à celle qui me semble la plus simple, la qualité générale sur une opération de services Web. Cette qualité peut être représentée par $\mathcal{Q}(s)$, s étant un service Web de $\mathbf{S}$. Ce critère de qualité devra être défini par chacune des méthodes, et devra permettre de trier les compositions par ordre de qualité. Il sera possible de parler de $\mathcal{Q}(c)$ et donc de choisir la composition la plus appropriée. Comme nous l'avons vu ce problème reste largement ouvert.

3.6 Objectif de la thèse

L'objectif de la thèse est de proposer une méthode dont le but est d'inventer des compositions que je viens de formaliser, à partir des connaissances de l'environnement Γ, ces connaissances étant généralement partielles.

Générer des compositions de services Web est un problème difficile, comme le montre l'état de l'art. La formalisation proposée dans ce chapitre explicite certaines caractéristiques des services Web en météorologie, comme le fait qu'une composition ne modifie pas l'environnement, ce qui facilite la composition de services Web. Néanmoins, les services Web en météorologie manipulent des messages très riches. Il est donc facile d'imaginer que le domaine des messages est infini, ce qui rend la planification traditionnelle inadéquate. Par ailleurs, la preuve de programme montre également ses limites. Elle est souvent indécidable, même si parfois elle fonctionne très bien.

Chapitre 4

Etude préliminaire de trois méthodes de composition : statiques, dynamiques et inductives

Ce chapitre vient répondre à la question : comment créer de nouvelles compositions à partir des connaissances sur les services Web que l'on possède. J'y présente donc la réalisation de trois prototypes. Les deux premiers prototypes ont été réalisés dans le but d'expérimenter différentes approches de la littérature. Pour simplifier leur mise en oeuvre, ces prototypes utilisent tous leur propre langage de description. J'ai choisi de me concentrer sur les fonctions Fg et Fv, laissant ainsi de côté la fonction Ft.

4.1 Méthode de composition statique

Tout d'abord, j'expérimente une approche basée sur les plans statiques. Cette méthode est une évolution du chaînage des services Web réalisé par SoDa. Cette évolution introduit une forme de fusion de données et une forme de choix. En revanche, elle n'est pas dynamique et ne peut pas prendre en compte de nouveaux services Web automatiquement.

4.1.1 Principe

Le principe de cette méthode se base sur la formalisation du chapitre précédent, et consiste à implémenter un langage permettant de programmer les compositions de services Web de façon statique. Ce langage incorpore une forme d'union et une forme de choix. Du fait que la composition est statique, elle permet de personnaliser l'union de messages à chaque étape de la composition.

Représentation des messages J'ai choisi de représenter les messages par un ensemble de paires : identifiant et valeur. De cette façon, chaque message est en fait une

fonction, soit $\mathbf{M} \in \mathcal{P}(\mathbf{ID} \times \mathbf{V})$ où **ID** est l'ensemble des identifiants et **V** est l'ensemble des valeurs. Les valeurs sont des données abstraites que la méthode ne permet pas de manipuler. Dans l'implémentation, ces valeurs sont des chaînes de caractères représentant parfois des entiers, parfois des nombres à virgule flottante et parfois des tableaux.

Manipulation des messages Comme la méthode est statique, elle permet à l'utilisateur de définir ses propres fonctions $\mathcal{U}$. Une telle fonction est définie à l'aide d'opérateurs permettant de manipuler des messages. De plus, pour cette méthode, l'union peut s'effectuer sur plusieurs messages. Pour ce faire, les opérateurs réalisant l'union utilisent un environnement Γ comportant l'ensemble des messages connus à un instant (état) donné et identifiés par des identifiants appartenant à **ID**. Le langage permet d'ajouter ou de modifier les éléments des messages. Le langage définissant les unions de message comporte principalement deux opérateurs : **bind** et **cst**. Le premier **bind** consiste à créer ou modifier la valeur d'un message avec une autre valeur existante. Le second consiste à créer une nouvelle valeur à partir d'une constante.

Contrôle de l'exécution de la composition Le contrôle de l'exécution de la composition est réalisé à partir des opérateurs **try**, **seq**, **new**, **call** et **skip**. L'opérateur **try** a comme paramètre deux sous-compositions, qui devraient transformer l'environnement de façon similaire. Cet opérateur essaie d'exécuter la première composition. Si celle-ci échoue pour une raison quelconque, alors la seconde est exécutée. L'opérateur **seq** réalise l'enchaînement de deux compositions, la première composition est exécutée puis la suivante. L'opérateur **new** crée un nouveau message vide dans l'environnement. L'opérateur **call** fait appel à un service Web afin de créer un nouveau message correspondant à la réponse du service. Enfin l'opérateur **skip** ne fait rien.

Syntaxe du langage Cette dernière est définie par l'expression suivante :

$$
\begin{array}{rcl}
c & ::= & \textbf{new}\ id_0 \\
 & | & \textbf{try}\ c_0\ c_1 \\
 & | & \textbf{call}\ f\ id_0\ id_1 \\
 & | & \textbf{bind}\ id_0\ id_1\ id_2\ id_3 \\
 & | & \textbf{cst}\ id_0\ id_1\ v \\
 & | & \textbf{seq}\ b_0\ b_1 \\
 & | & \textbf{skip}
\end{array}
$$

4.1.2 Sémantique dans le cas idéal

Je présente dans cette section la sémantique dans le cas idéal, c'est-à-dire dans le cas où tout se déroule sans message d'erreur.

Sémantique de l'opérateur bind L'opérateur **bind** comporte quatre paramètres qui sont tous des identifiants. Néanmoins id_0 et id_2 identifient des messages de l'environnement alors que id_1 et id_3 identifient des valeurs dans les messages id_0 et id_2. Cet opérateur va donc copier la valeur id_3 du message id_2 dans le message id_0 avec l'identifiant id_1.

$$\frac{\Gamma(id_2)(id_3) = v}{\Gamma \vdash \textbf{bind}\ id_0\ id_1\ id_2\ id_3 \rightarrow \Gamma\,[id_0 \mapsto \Gamma(id_0)\,[id_1 \mapsto v]]} \tag{4.1}$$

Sémantique de l'opérateur cst Cet opérateur prend trois paramètres, un identifiant de message id_0, un identifiant de valeurs id_1 et une valeur immédiate v. Dans le cas idéal, sa sémantique est la suivante :

$$\frac{}{\Gamma \vdash \textbf{cst}\ id_0\ id_1\ v \rightarrow \Gamma\,[id_0 \mapsto \Gamma(id_0)\,[id_1 \mapsto v]]} \tag{4.2}$$

Sémantique de l'opérateur try L'opérateur **try** utilise deux paramètres, chacun d'eux étant une expression de composition. Dans le cas idéal, la sémantique de cet opérateur est équivalente à l'évaluation de la première composition c_0, ce qui donne :

$$\frac{\Gamma \vdash c_0 \rightarrow \Gamma'}{\Gamma \vdash \textbf{try}\ c_0\ c_1 \rightarrow \Gamma'} \tag{4.3}$$

L'opérateur call Cet opérateur utilise trois paramètres, le premier f est l'opération de service Web à réaliser, le second id_0 est l'identifiant du message qui doit être utilisé pour appeler le service Web. Le dernier paramètre id_1 est l'identifiant du message qui sera reçu en retour.

$$\frac{(\Gamma(id_0), v) \in f \& v \in \mathbf{M}}{\Gamma \vdash \textbf{call}\ f\ id_0\ id_1 \rightarrow \Gamma\,[id_1 \mapsto v]} \tag{4.4}$$

L'opérateur seq Cet opérateur utilise deux paramètres qui sont deux expressions de composition. Ces deux expressions sont évaluées l'une après l'autre comme le définit la sémantique suivante :

$$\frac{\Gamma \vdash b_0 \rightarrow \Gamma' \quad \Gamma' \vdash b_1 \rightarrow \Gamma''}{\Gamma \vdash \textbf{seq}\ b_0\ b_1 \rightarrow \Gamma''} \tag{4.5}$$

L'opérateur skip Cet opérateur ne prend pas de paramètre et il est défini de la façon suivante :

$$\frac{}{\Gamma \vdash \textbf{skip} \rightarrow \Gamma} \tag{4.6}$$

L'opérateur new Cet opérateur comporte un paramètre représentant l'identifiant d'un nouveau message. Il est défini par :

$$\frac{}{\Gamma \vdash \textbf{new}\ id \rightarrow \Gamma\,[id \mapsto \emptyset]} \tag{4.7}$$

4.1.3 Sémantique dans le cas réel

Dans le cas réel, comme nous l'avons déjà vu, des erreurs peuvent survenir. Pour prendre en compte ces erreurs et pouvoir les propager, je propose d'ajouter un élément $Fail$ qui représente une erreur dans le déroulement de la composition. L'exécution de la composition peut donc finir par un environnement Γ ou une erreur $Fail$. Par ailleurs, les opérateurs pouvant provoquer des erreurs sont **seq**, **call**, **try**, **bind** et **cst**. Voici les règles à ajouter pour prendre en compte les erreurs.

L'opérateur call Cet opérateur échoue lorsque l'appel de l'opération du service Web aboutit à un message d'erreur. La règle à ajouter est la suivante :

$$\frac{(\Gamma(id_0), v) \in f \& v \in \textbf{Erreur}}{\Gamma \vdash \textbf{call}\ f\ id_0\ id_1 \rightarrow Fail} \tag{4.8}$$

L'opérateur seq Cet opérateur échoue si l'une des deux expressions de composition échoue. Ceci se traduit par les règles suivantes :

$$\frac{\Gamma \vdash b_0 \rightarrow Fail}{\Gamma \vdash \textbf{seq}\ b_0\ b_1 \rightarrow Fail} \tag{4.9}$$

$$\frac{\Gamma \vdash b_0 \rightarrow \Gamma' \quad \Gamma' \vdash b_1 \rightarrow Fail}{\Gamma \vdash \textbf{seq}\ b_0\ b_1 \rightarrow Fail} \tag{4.10}$$

L'opérateur try Cet opérateur échoue seulement si les deux expressions de composition associées échouent. Il faut donc ajouter deux nouvelles règles, la première définissant le cas où l'une des deux compositions échoue et la dernière dans le cas où les deux compositions échouent.

$$\frac{\Gamma \vdash c_0 \rightarrow Fail \quad \Gamma \vdash c_1 \rightarrow \Gamma'}{\Gamma \vdash \textbf{try}\ c_0\ c_1 \rightarrow \Gamma'} \tag{4.11}$$

$$\frac{\Gamma \vdash c_0 \rightarrow Fail \quad \Gamma \vdash c_1 \rightarrow Fail}{\Gamma \vdash \textbf{try}\ c_0\ c_1 \rightarrow Fail} \tag{4.12}$$

L'opération bind Cette opération échoue si l'un des identifiants de message ne référence aucun message valide ou si l'identifiant de valeur à copier n'existe pas.

$$\frac{\forall m.(id_3, m) \notin \Gamma}{\Gamma \vdash \textbf{bind}\ id_0\ id_1\ id_3\ id_4 \rightarrow Fail} \tag{4.13}$$

$$\frac{\forall m.(id_0, m) \notin \Gamma}{\Gamma \vdash \textbf{bind}\ id_0\ id_1\ id_3\ id_4 \rightarrow Fail} \tag{4.14}$$

$$\frac{\forall v.(id_3, m) \in \Gamma\ \&\ (id_4, v) \notin m}{\Gamma \vdash \textbf{bind}\ id_0\ id_1\ id_3\ id_4 \rightarrow Fail} \tag{4.15}$$

L'opérateur cst Cet opérateur échoue seulement si le message de destination n'existe pas.

$$\frac{\forall m.(id_0, m) \notin \Gamma}{\Gamma \vdash \textbf{cst}\ id_0\ id_1\ v \rightarrow Fail} \tag{4.16}$$

4.1.4 Initialisation et Exécution

Pour pouvoir exécuter une composition, il faut pouvoir définir les paramètres d'entrée de cette composition. Ces paramètres peuvent être définis à l'aide d'un nouvel opérateur **main ... return** dont la syntaxe est la suivante :

$$e := \textbf{main}\ id_0\ c\ \textbf{return}\ id_1 \tag{4.17}$$

Cet opérateur consiste à initialiser le message id_0 avec le message m fourni par l'utilisateur et à renvoyer le message id_1 à l'utilisateur. Dans le cas où id_1 n'existe pas ou que la composition échoue, alors $Fail$ sera renvoyé à l'utilisateur.

La sémantique est donc la suivante :

$$\frac{\{(id_0, m)\} \vdash c \mapsto Fail}{m \vdash \textbf{main}\ id_0\ c\ \textbf{return}\ id_1 \rightarrow Fail} \tag{4.18}$$

$$\frac{\{(id_0, m)\} \vdash c \mapsto \Gamma \quad \forall m'.(id_1, m') \notin \Gamma}{m \vdash \textbf{main}\ id_0\ c\ \textbf{return}\ id_1 \rightarrow Fail} \tag{4.19}$$

$$\frac{\{(id_0, m)\} \vdash c \mapsto \Gamma \quad (id_1, m') \in \Gamma}{m \vdash \textbf{main}\ id_0\ c\ \textbf{return}\ id_1 \rightarrow m'} \tag{4.20}$$

4.1.5 Implantation

Comme cette méthode est une évolution de composition de services Web de SoDa, son implémentation utilise les services de SoDa. Ces derniers ne suivent pas la définition des services Web car ils n'utilisent pas le protocole SOAP. A la place de ce protocole, les services SoDa utilisent les fonctionnalités GET et POST du protocole HTTP. Les messages sont donc bien des listes de couples $\langle id, v \rangle$ mais sont transmis en HTTP. Le résultat des services SoDa est fourni sous la forme d'un flux XML, dont le format a été pré-établi par [105].

Le langage défini a été traduit en XML. Par exemple, les `seqb` sont représentés par des listes de `input_binding` et `output_binding`. Chacun d'eux comporte des attributs comme `from`, `to` ou `value`.

Chaque service est décrit par son URL (Unique Resource Link) et la méthode employée pour envoyer le message : GET ou POST. De plus, comme les services ne sont pas standards, j'ai choisi de rendre les compositions homogènes au service, en les appelant à l'aide du même protocole. La composition renvoie le résultat sous la forme d'un flux de données XML, au format utilisé par les services de SoDa.

4.1.6 Discussion

J'ai développé un langage s'inspirant des langages de description de processus pour décrire les compositions. Ce langage est basé sur le XML et l'application SoDa. Mon prototype est limité à la composition automatique, et au remplacement d'un service

par un autre, lorsque le premier renvoie une réponse inattendue. Néanmoins, il donne un bon aperçu des capacités de cette approche et des difficultés à surmonter.

Ce type de méthode peut répondre à tous les besoins liés à la composition automatique lorsque le langage de composition utilisé est suffisamment expressif, comme par exemple BPEL4WS [12, 13]. De plus, il dispose de moyens pour représenter les choix alternatifs de plans, c'est-à-dire qu'il permet de définir plusieurs plans pour accomplir une même tâche, permettant ainsi d'utiliser un plan alternatif lorsque le premier choix de plan a échoué. En revanche, cette approche n'est pas complètement adaptative car elle ne prend pas en compte les changements du contexte internet lorsqu'on ajoute un service Web car les plans sont pré-construits à la main et restent statiques. Par exemple, si l'on dispose d'un service Web fournissant une série temporelle de températures en degrés Farenheit, et que par la suite, on ajoute un service qui se charge de convertir une série temporelle en degrés Farenheit en une série temporelle en degrés Celsius, cette méthode ne pourra pas fournir des séries temporelles en degrés Celsius car la composition de ces deux services n'a pas été prévue à l'origine. Ceci illustre simplement la différence entre les approches de composition statique de celles de composition dynamique.

L'approche des plans statiques est intéressante car elle est capable de décrire tous les flux de données que l'on pourrait souhaiter. L'inconvénient majeur est qu'elle ne prend pas facilement en compte les nouveaux services Web ; lors d'un ajout de service Web, un expert est requis pour re-écrire tous les modèles de plan. Ce processus de compréhension et d'intégration des services Web est source d'erreurs, prend du temps et peut être difficile [23].

4.2 Méthode de composition dynamique basée sur des graphes de dépendance d'états

Compte tenu des limitations du prototype précédent, je propose ici un prototype qui génère les compositions de façon automatique et adaptative. Je commence par présenter son principe, puis je montre son implémentation pour finir sur une évaluation informelle sous la forme d'une discussion.

Cette méthode de composition dynamique s'inspire des méthodes de planification. Pour répondre à une requête de l'utilisateur, elle utilise la description des entrées et des sorties des services Web pour construire un graphe, qui symbolise les transformations possibles de données, où les noeuds du graphe sont les états du monde et les transitions sont les actions, à la manière de [44]. Ensuite, un programme énumère les chemins dans ce graphe qui relient un état initial représentant les entrées, fournies par l'utilisateur, et un état final, représentant les sorties souhaitées par l'utilisateur. Finalement, le moteur trie les chemins suivant un critère de qualité prédéfini, puis déduit de ces chemins, des compositions qu'il peut ensuite exécuter. La méthode commence par exécuter le chemin de meilleure qualité, puis vérifie le résultat obtenu. Si le résultat est satisfaisant, alors le programme renvoie le résultat à l'utilisateur, sinon il essaie d'exécuter la composition suivante. Dans cette méthode le résultat de la composition sera considéré comme satisfaisant lorsqu'il comportera tous les éléments souhaités par l'utilisateur. Par exemple, si l'utilisateur demande une liste d'éclairement, alors la composition est validée dès lors que cette liste est présente, quel que soit son contenu. Le résultat de la composition suivante est combiné avec le résultat de l'exécution précédente ; le résultat ainsi obtenu est à nouveau analysé pour déterminer s'il faut

continuer l'amélioration ou renvoyer le résultat à l'utilisateur.

La première section explique la représentation des services Web, puis celle des états du monde, ainsi que la construction du graphe de dépendance des états. Nous décrivons ensuite l'algorithme qui génère l'ensemble des séquences d'actions équivalentes. Puis, je présente les critères de qualité ainsi que la méthode de tri des séquences d'actions. Enfin, je donne la méthode d'union et d'évaluation des résultats d'exécution des compositions.

4.2.1 Représentation des services Web

Chaque service Web est décrit comme une action fournissant un ensemble de paramètres de types prédéfinis à partir d'un autre ensemble de paramètres de types prédéfinis. Soit $\mathbf{T}$ l'ensemble des types de paramètres possibles et $\mathbf{F}$ l'ensemble des identifiants de services Web, alors un service Web est défini par le tuple (F, t_i, t_f) appartenant à $\mathbf{F} \times \mathcal{P}(\mathbf{T}) \times \mathcal{P}(\mathbf{T})$, avec t_i représentant les pré-conditions de l'action et t_f étant les post-conditions.

Le choix de cette représentation est lié au WSDL [110] qui définit les services Web de façon standard en terme d'entrées et de sorties. Chacune des parties des entrées et des sorties du service Web est typée.

4.2.2 Abstraction de l'environnement

La méthode définit les états du monde comme l'union des types des données fournies ou attendues par l'utilisateur. Soit $\mathbf{T}$ l'ensemble de tous les types de données possibles, alors l'ensemble $\mathbf{S}$ appartient à $\mathcal{P}(T)$. Chaque type d'un même état représente une donnée possible à obtenir dans cet état. Par exemple, un état composé des types *géopoint* et *période* indique que les données atteignables sont obligatoirement de ces deux types.

4.2.3 Construction du graphe de dépendance

Pour construire le graphe de dépendance, chaque description de services Web est analysée pour créer l'ensemble des relations $\mathbf{R}$ reliant les états. Comme nous l'avons vu en planification dans le chapitre 2, les relations appartiennent à $S \times A \times S$. Pour déterminer l'existence d'une relation, il faut vérifier les pré-conditions et les post-conditions. Les pré-conditions sont respectées lorsque les types d'entrée d'un service Web sont inclus dans l'état courant. Les post-conditions sont réalisées lorsque le nouvel état est égal à l'ensemble des types de sortie. Autrement dit, pour chaque tuple (f, s_i, s_f) appartenant à $\mathbf{F} \times \mathbf{S} \times \mathbf{S}$, la méthode vérifie que les pré-conditions et les post-conditions sont réalisées, puis crée un arc dans le graphe des états reliant s_i et s_f représentant la transformation que peut réaliser le service Web f. Par exemple, un service Web qui convertit une température de Celsius à Farenheit est représenté par un arc qui relie les noeuds *température en Celsius* à *température en Farenheit.*

4.2.4 Génération des séquences d'actions équivalentes

Le graphe est donc constitué de noeuds, représentant les ensembles de paramètres typés possibles et d'arcs représentant les actions possibles pour passer d'un ensemble à un autre.

Chaque chemin dans le graphe représente une composition obéissant à des contraintes fonctionnelles nécessaires. Néanmoins, se baser uniquement sur ces chemins n'est pas efficace et notre méthode utilise la requête de l'utilisateur pour ne sélectionner que les chemins intéressants vis-à-vis de cette requête. Pour ce faire, la méthode cherche les noeuds dans le graphe représentant les données pouvant être fournies par la requête. Puis, elle recherche les noeuds pouvant fournir les données souhaitées par l'utilisateur. Enfin, la méthode cherche les chemins sans boucle reliant ces noeuds. Un chemin sans boucle est un chemin passant au plus une fois dans un arc donné.

Il existe, pour trouver ces chemins, des algorithmes qui terminent et sont *complets*. Un algorithme complet dans notre cas est un algorithme qui trouve toutes les solutions existantes. Il est formalisé par : pour tout X, si X est un chemin sans boucle et valide, alors l'algorithme trouve X comme solution. Je présente un de ces algorithmes dans la section suivante.

Algorithme de génération de toutes les séquences d'actions équivalentes Il existe deux façons standards de décrire les graphes. La première est une représentation basée sur des tableaux et la seconde est basée sur la liste d'adjacence [100].

Pour réaliser mon algorithme de recherche de chemin, j'ai choisi de représenter un graphe par des listes d'adjacence. Cette représentation liste, pour chaque noeud, les noeuds adjacents, autrement dit, les noeuds accessibles directement par un arc. Un graphe g est donc représenté par la liste V de ses noeuds. Chaque noeud $v \in V$ comporte la liste des arcs $e \in Adj(v)$ menant à ses noeuds adjacents. Chaque arc peut être marqué à l'aide d'une fonction $mark(g, e)$ et démarqué à l'aide de $unmark(g, e)$. L'état de la marque peut être lu grâce à $is\_mark(g, v)$ retournant vrai si v est marqué et faux sinon.

L'algorithme suivant génère la liste P de tous les chemins entre un sommet de départ v_{begin} et un sommet d'arrivée v_{end}. Pour cela, partant de v_{begin}, il considère tous les arcs de sa liste d'adjacence. Pour chacun des arcs de cette liste, il va chercher tous les chemins reliant le noeud d'arrivée de l'arc à v_{end} sans réutiliser l'arc. Ces chemins sont ensuite concaténés avec l'arc courant de manière à générer l'ensemble de tous les chemins passant par cet arc. Cette génération est faite de manière récursive.

```
P : ensemble de chemins ;
p : chemin ;
v_begin : noeud ;
v_end : noeud ;
define follow (v, p, P)
    (marque le noeud pour ne plus y repasser)
    mark(v) ;
    (ajoute le noeud courant au chemin)
    p.push(v) ;
    (pour tous les noeuds adjacents du noeud courant)
    foreach i in Adj(v) :
        (si le noeud adjacent est celui de fin alors on ajoute
        le chemin à la liste des chemins, sinon on poursuit la recherche)
        if i == v_end then
            p.push(i) ;
            P.add(P) ;
```

```
            p.pop(i) ;
        else
            (si le noeud adjacent n'est pas marqué alors on poursuit la
            recherche, sinon on abandonne ce chemin)
            if not is_mark(i) then
                follow(i, p. P) ;
            endif
        endif
    endfor
    (on a fini la cherche pour le noeud courant)
    p.pop(v)
    (il n'est plus marqué)
    unmark(v)
enddef

follow (v_begin, p, P) ;
return P ;
```

Une fois l'algorithme terminé, nous disposons de l'ensemble des chemins sans boucle qui relient les entrées fournies aux sorties souhaitées. Il faut ensuite convertir ces chemins dans un langage de composition. Pour ce prototype, ce langage est le même que celui du prototype précédent 4.1 et utilise donc uniquement les séquences et les assignations. Je montrerai dans l'exemple en fin de chapitre comment sont traduits ces chemins en composition.

4.2.5 Critère de qualité des chemins

Une fois les compositions possibles établies, il faut les trier par ordre de qualité. Nous devons donc définir un critère de qualité permettant ce tri.

La notion de qualité de données est très subjective et dépend parfois des besoins, comme nous l'avons vu dans le chapitre 2. Dans mon cas, j'ai retenu un critère simple, qui est une valeur de qualité comprise entre 0 et 1. Plus cette valeur est proche de 1, plus la composition est de bonne qualité. Un concepteur de services Web attribue à chacun d'eux une valeur de qualité de service entre 0 et 1. Je définis le critère de qualité d'une composition comme le produit des qualités de chacun des services Web qui la composent. Les compositions peuvent alors être triées par ordre décroissant de qualité.

4.2.6 Validation des résultats

Une fois les compositions triées, nous exécutons la première et nous devons la valider. La validation est réalisée en comparant les types des données obtenues après l'exécution de la composition et les types des données souhaitées par l'utilisateur. Lorsque l'ensemble des types souhaités est inclus dans l'ensemble des types des données du résultat, alors la composition est validée. Par exemple, si l'utilisateur souhaite obtenir une liste d'éclairement, et que le résultat obtenu après exécution de la composition est une liste de températures alors le résultat est invalide. En revanche, si le résultat obtenu comporte une liste de température et une liste d'éclairement, le résultat est validé. Cette méthode ne prend pas en compte le contenu des données, elle compare uniquement les types des données.

4.2.7 Union des résultats

Dans le cas où la composition n'est pas validée, la composition suivante est exécutée et son résultat fusionné avec la précédente. Dans ce prototype, l'union est réalisée simplement par le remplacement de l'ancien résultat par le nouveau. Ceci peut ne pas être considéré comme une union à proprement parler, mais elle est clairement suffisante pour analyser notre méthode.

4.2.8 Discussion

Facilité de mise en oeuvre de la méthode Cette approche fournit une méthode raisonnablement facile pour combiner les services Web, puisque la traduction des descriptions standards des entrées/sorties (exprimées en WSDL) en pré-/post-conditions est simple.

Une forte abstraction Elle s'appuie sur le fait que les ensembles des types décrivant les messages d'entrées et les messages de sorties d'un service Web sont une abstraction de ce qu'il réalise. Autrement dit, les entrées et les sorties du service Web représentent d'une certaine façon ce que réalise le service Web. Cette représentation n'est pas complète, elle ne comporte pas, par exemple, les attributs non-fonctionnels (par exemple, le nom de l'algorithme utilisé), ou encore la manière dont vont être utilisées les données d'entrée. Par exemple, deux services Web effectuant la même transformation de données dans le graphe, peuvent en réalité faire appel à des opérations différentes. Un service Web qui additionne a et b, et un service Web qui calcule a fois b, ne peuvent être différenciés car ils ont les mêmes pré-/post-conditions qui sont basées pour la méthode sur les types des entrées/sorties. On remarque également que la description des pré-conditions et post-conditions a un impact important sur les compositions qu'il sera possible de faire ainsi que leur justesse. Plus la description des pré-/post-conditions est précise, plus la proportion de compositions valides sera grande. En contre-partie, des compositions valides seront éliminées, et vice-versa, plus les descriptions sont grossières et peu détaillées, plus la proportion de compositions fausses qui seront générées sera grande. Par exemple, prenons deux services Web le premier, nommé F, s'appliquant sur un entier naturel et donnant en résultat un entier naturel, et le second, nommé G, s'appliquant sur un entier positif et fournissant en sortie un entier naturel. Si l'on choisit de décrire ces 2 services Web grossièrement en ignorant l'existence des entiers positifs, alors F et G sont décrits de la même manière et la sortie de F peut être utilisée en entrée de G. Dans certains cas cette composition est invalide, néanmoins elle reste valide lorsque la sortie de F est positive. Dans le cas de descriptions grossières, c'est-à-dire ici très abstraites par rapport à la fonctionnalité des services Web, on a donc créé des compositions fausses sans éliminer de compositions valides. Par ailleurs, si on choisit de décrire de façon rigoureuse ces deux fonctions alors G ne peut plus être appliquée sur le résultat de F et on élimine toutes les compositions de F et de G où les sorties de F sont positives. Dans ce cas on a une description plus détaillée, mais qui élimine des compositions valides.

Des algorithmes alternatifs Etant donné que l'ensemble des arcs est dénombrable, l'ensemble des états l'est aussi (leur nombre étant au maximum deux fois supérieur au nombre d'actions). Ce problème est donc solvable par des méthodes de planification classiques. Il est donc tout à fait possible d'utiliser des méthodes issues

de la planification. Pour cela, il faudrait traduire les descriptions des services Web dans des langages utilisés pour la planification tel que PDDL [38]. Par ailleurs, pour trouver un enchaînement d'actions possibles, l'algorithme de Dijkstra peut être utilisé.

Néanmoins, dans ces deux cas, l'un des principaux besoins exprimés par les météorologues étant la complétion des résultats, l'obtention de l'unique composition, fournie par les méthodes de planification classiques ou par l'algorithme de Dijkstra, n'est pas satisfaisante. Nous devons donc générer plusieurs enchaînements d'actions équivalents afin de pouvoir effectuer l'union des résultats de leur exécution pour atteindre un bon niveau de complétion des données.

Limite de notre approche Cette approche n'est pas capable d'utiliser conjointement deux services Web pour calculer les entrées d'un troisième compte tenu que les chemins d'un graphe sont purement séquentiels. Un moyen pour contourner ce manque de parallélisme inhérent aux graphes est de créer des services Web virtuels qui représentent l'exécution parallèle de plusieurs services. Cette exécution concurrente est représentée par une transition qui aurait pour entrée l'union des entrées des services Web utilisés et pour sortie l'union des sorties de ces services Web. Cette amélioration s'accompagne d'un accroissement exponentiel du nombre des états et des transitions du graphe (2^n, où n est le nombre de services Web).

Avantage de notre méthode de composition Ce prototype présente l'avantage de prendre en considération de nouveaux services Web contrairement à la méthode de composition statique précédente.

4.3 Méthode d'induction Prolog

Le troisième prototype est basé sur la logique du premier ordre. Il tente de répondre à la question *Existe-t-il une séquence de services Web reliant les entrées fournies par l'utilisateur aux sorties souhaitées* à partir d'un certain nombre d'énoncés supposés vrais, ou axiomes. Contrairement à la précédente, cette méthode permet d'exprimer les relations entres les entrées et les sorties d'un service Web par une fonction.

Dans cette section, je commence par présenter Prolog et son principe d'inférence, puis je décris la représentation des messages utilisés en entrée et en sortie par les services Web, je donne ensuite la manière de représenter des services Web en Prolog, je montre la manière de générer les compositions et je discute de qualité et de complétion des résultats avant de conclure.

4.3.1 Présentation de Prolog

Prolog a été créé par Alain Colmerauer et Philippe Roussel vers 1972 [86]. Leur but était de faire un langage de programmation qui permettait d'utiliser l'expressivité de la logique au lieu de définir pas à pas la succession d'instructions que doit exécuter un ordinateur. Le nom Prolog est un acronyme de PROgrammation LOGique.

Prolog est utilisé dans de nombreux programmes d'intelligence artificielle et dans le traitement de la linguistique par ordinateur en partuculier ceux concernant les langages naturels. Sa syntaxe et sa sémantique sont considérées comme très simples et claires.

Prolog est basé sur le calcul des prédicats du premier ordre ; cependant, il est restreint dans sa version initiale à n'accepter que les clauses de Horn (cf. 2.5). Les versions plus récentes de Prolog acceptent des prédicats plus complexes, notamment avec le traitement de la négation par l'échec. Dans notre cas, nous utiliserons uniquement des clauses Horn.

L'exécution d'un programme Prolog consite à prouver une affirmation par unification successive et récursive des différentes données de la base de connaissances. Une des particularités de Prolog est que l'on peut construire une base de connaissances dans un ordre indéterminé. Prolog peut ensuite résoudre des séries de problèmes logiques relatifs à une telle base de connaissances.

4.3.2 Description des messages

Les messages d'entrée et de sortie des services Web dans cette méthode sont représentés par une liste Prolog de prédicats. Par exemple, un message comportant un *geopoint* et une *period* se représente par la liste des deux prédicats suivants :

```
[geopoint(48,2),period(20080101,20080201)]
```

Dans cet exemple, le geopoint est défini à la latitude 48 et la longitude 2 ; la période est entre les deux dates données. Il est possible si nécessaire de définir des paramètres sans valeur.

4.3.3 Description des services Web

Chaque service Web est traduit par un ou plusieurs axiomes de la logique du premier ordre qui relie ses entrées à ses sorties. Dans le cas général, un service Web se définit par un axiome affirmant que *le service Web nommé A ayant pour entrée I et pour sortie O existe si l'on peut relier I et O par une relation prédéfinie* [45].

Prenons l'exemple, d'un service Web nommé service3 qui fournit une série temporelle de température en degrés Farenheit en un lieu donné. On le décompose en axiomes de la manière suivante :

```
(* trouve la première occurence de X dans la liste [A|B] *)
find(X, [A|B]) :-
  X = A, !;
  find(X, B).

(* macro définissant une liste de températures sur une période donnée *)
getseqtemperature(Period, Unit, Geopoint, [A]) :-
  Period = period(X, X),
  A = temperature(X, Unit, Geopoint).
getseqtemperature(Period, Unit, Geopoint, [A|B]) :-
  Period = period(Begin, End),
  Begin < End,
  succ(Begin, Next),
  getseqtemperature(period(Next, End), Unit, Geopoint, B),
  A = temperature(Begin, Unit, Geopoint).

(* définit un service Web nommé service3 qui fournit une
```

```
   liste de température en degrés farhenheit *)
service(service3, I, O) :-
  find(period(B,E), I),
  find(geopoint(X,Y), I),
  getseqtemperature(period(B,E), farenheit, geopoint(X,Y), O),!.
```

Les premiers axiomes `find(X, L)` définissent l'opération de recherche d'un paramètre `X` dans la liste `L`. Les axiomes `getseqtemperature` créent une liste qui représente une série temporelle de température. Enfin l'axiome `service3` représente le service Web et les contraintes liant ses entrées et ses sorties. On lance le processus d'unification sur le programme Prolog. Comme il est dit dans la section 4.3.1, s'il existe une solution, alors l'unification de Prolog la trouve en un temps fini. S'il n'existe pas de solution, il se peut que Prolog ne termine pas. Dans notre cas, comme nous cherchons à obtenir plusieurs solutions (de manière à exhiber plusieurs compositions possibles satisfaisant la requête), il se peut que l'unification ne termine pas alors que nous avons déjà obtenu un ensemble de solutions. L'unification permet de construire toutes les solutions possibles. Ainsi, sur l'entrée :

```
[period(20080101, 20080103), geopoint(x,y)]
```

nous lançons la requête :

```
service(service3, [period(20080101, 20080103), geopoint(x,y)], O)
```

et obtenons la sortie :

```
[temperature(20080101, farenheit, geopoint(x,y)),
 temperature(20080102, farenheit, geopoint(x,y)),
 temperature(20080103, farenheit, geopoint(x,y))]
```

Cette sortie représente toutes les valeurs obtenues sur la période 2008-01-01, 2008-01-03 pour un lieu quelconque. Cette description montre bien la construction de la série temporelle de température. Notons que la définition des prédicats avec leurs paramètres doit être pré-établie : dans cet exemple, `temperature` comporte uniquement 3 paramètres : une date, une unité et une localisation, mais on pourrait imaginer y rajouter l'altitude.

4.3.4 Réalisation de Fg

Je définis une composition en Prolog par l'affirmation suivante : *Un service composé [A,B] ayant pour entrée I et pour sortie O existe s'il existe un service A reliant l'entrée I à E et un service composé B reliant E à O.* Cette affirmation s'écrit en Prolog par les axiomes suivants, où `servicec` est une composition de service et `service` est un service atomique :

```
servicec([], X, X).
servicec([A|B], I, O) :-
  service(A, I, E),
  servicec(B, E, O).
```

Pour générer toutes les compositions possibles correspondant à une requête de l'utilisateur, j'utilise le principe d'unification de Prolog sur les règles Prolog de composition précédentes et sur les axiomes représentant les services.

Dans la requête suivante, la première ligne définit la forme requise pour les sorties et la seconde ligne demande à rechercher les compositions X produisant cette sortie si les entrées sont `period` et `geopoint` donnés.

```
getseqtemperature(period(20080101,20080110),
  celsius, geopoint(48,2), O),
servicec(X, [period(20080101, 20080110), geopoint(48,2)], O)
```

Après unification, dans notre cas, comme il existe des services composés qui relient les entrées aux sorties de la façon souhaitée, on obtient les solutions satisfaisant la requête suivante :

```
O = ...
X = [service1] ;
O = ...
X = [service3, service2] ;
false.
```

Notons que dans notre exemple, l'arbre de recherche étant fini, l'unification termine. Elle rend `false` pour indiquer, quand on lui demande de trouver une autre solution, qu'il n'en existe plus.

Ces solutions sont alors utilisées pour générer les compositions de services Web correspondantes. Ces compositions seront ensuite exécutées et leur résultat renvoyé à l'utilisateur.

4.3.5 Qualité et complétion des résultats

Le prototype dans sa version courante ne gère pas la qualité des services Web. En effet, actuellement, les solutions sont analysées les unes après les autres dans l'ordre dans lequel Prolog les trouve. Pour prendre en compte la qualité, il serait possible de procéder à un tri des solutions par qualité décroissante. Dans le cas général, le nombre de solutions étant possiblement infini, il faudrait forcer l'algorithme à terminer sur certains critères comme, par exemple, limiter la recherche à des chemins impliquant moins de 10 services Web. On peut alors employer la méthode de qualité utilisée dans le prototype précédent.

On rencontre le même problème pour la complétion des résultats, les compositions arrivent sans ordre de qualité et leur possible infinité nous empêche de les trier. L'union n'est donc pas réalisée dans ce prototype.

4.3.6 Discussion

Les règles de composition des services Web définies par cette méthode sont plus expressives que celles de l'approche précédente. En effet, celle-ci se contentait de mettre en relation des entrées avec des sorties en ne se basant que sur les types des valeurs, alors que la méthode basée sur Prolog explicite les transformations de ces valeurs. Prenons l'exemple du `service3` précédent, nous constatons que sa sortie a pris en compte les valeurs des bornes de la période fournie en entrée, ce qui serait

impossible pour la méthode précédente. Cette méthode, en intégrant bien la façon dont les services Web transforment les données, permet de différencier des services qui seraient pris comme identiques par la méthode précédente.

Notons que cette approche, tout comme la précédente, ne permet pas de combiner deux services Web afin d'en utiliser un troisième.

Par ailleurs, contrairement à l'approche précédente qui recherchait des chemins sans boucle, cette approche n'élimine pas les *boucles*. Certaines preuves, comme par exemple, une preuve appliquant la même séquence d'axiomes en boucle [10], seront impossibles à obtenir. Notons que ce problème est susceptible de se produire fréquemment. Par exemple, si nous disposons d'une conversion d'une unité vers une autre et de sa réciproque, la solution alternera une infinité de fois le premier et le second service. Une solution pour résoudre ce problème serait de limiter le nombre de solutions en bornant la longueur de la séquence de services Web. Ce faisant, la méthode n'explore plus toutes les solutions, mais cette limitation me semble malgré tout judicieuse. Compte-tenu des latences du réseau Internet et du temps d'exécution des services Web, il devient primordial de ne pas utiliser un grand nombre de services Web afin de limiter les temps de réponse. La réponse à une requête à travers Internet est de l'ordre de 50 fois plus lente que si elle était appelée en réseau local. Ainsi demander une combinaison d'une dizaine de services Web débouche sur un temps d'exécution rédhibitoire. Une autre solution consisterait à limiter le temps de calcul en interrompant le processus de recherche de solution. De la même façon que la solution précédente, la méthode ne trouvera pas toutes les solutions.

L'approche basée sur Prolog, comme les approches de preuve de théorème, permet de donner une première approximation de la manière dont le service Web fonctionne. Cette approche reste difficile à rendre opérationnelle puisqu'elle nécessite, pour chaque service Web, l'ajout d'informations de transformation des données qui ne sont actuellement pas disponibles chez les fournisseurs de services Web. Notons que ces informations peuvent être complexes à fournir même si OWL-S [74] et WSMO [111] incitent déjà les fournisseurs à une définition plus sémantique des fonctions offertes. L'adoption de cette approche implique aussi des changements dans les techniques de conception actuelles comme, par exemple, l'adoption de standard de description comme OWL-S.

Plusieurs limites apparaissent dans cette approche. La définition des axiomes reste très complexe pour les non-programmeurs. Il faudrait proposer un langage de description des services Web plus adaptés au domaine des services Web et de la météorologie utilisant une représentation des données proche des arbres XML.

4.4 Résultats de cette étude

Le prototype basé sur les graphes montre que les méthodes de composition s'appuyant sur la planification automatique peuvent être limitées afin de garantir leur terminaison mais, à cause de cette limitation, elles sont susceptibles de proposer des solutions invalides vis-à-vis de la requête de l'utilisateur.

Le prototype d'induction Prolog montre que les méthodes de composition basées sur la preuve de programme deviennent indécidables lorsque l'expressivité du langage croît [58], que ce soit pour une preuve de théorème ou pour une synthèse automatique de programme. Par conséquent, elles nécessitent généralement l'intervention humaine.

Chapitre 5

Méthode hybride de composition

Je présente dans ce chapitre une méthode originale de composition automatique et adaptative des services Web. Cette méthode s'appuie sur deux principales conclusions tirées des précédentes méthodes de compositions et de l'état de l'art. En conséquence, dans notre méthode hybride, nous utilisons la planification pour générer des compositions en nous limitant aux compositions n'utilisant pas deux fois le même service Web et nous vérifions la validité des ces compositions par des techniques de preuve de programmes.

5.1 Les différentes phases du prototype

Le prototype se décompose selon les 7 étapes suivantes :

i. il établit la liste des compositions **Cs** possibles à partir de la description des types des données fournies et souhaitées par l'utilisateur et des types des entrées et sorties des services Web ;

ii. il valide de façon théorique ces compositions à l'aide d'un processus de vérification qui utilise la description sémantique des services Web et de la requête de l'utilisateur ;

iii. il élimine les compositions non validées à l'étape précédente ;

iv. il trie les compositions en fonction de leur qualité présumée ;

v. il exécute la première composition et combine son résultat avec les résultats précédemment obtenus, le premier résultat est combiné avec un résultat vide ;

vi. il compare le résultat réel obtenu par l'exécution de la composition avec le résultat théorique de la requête que souhaite réaliser l'utilisateur afin de déterminer si le résultat est complet et correspond au souhait de l'utilisateur ;

vii. dans le cas où le résultat n'est pas validé dans l'étape précédente et que des compositions sont encore disponibles $cs \in Cs$, il recommence, à partir de l'étape iv., à exécuter la prochaine composition dans la liste Cs. Dans les autres cas, il renvoie le résultat à l'utilisateur.

Les éléments permettant de réaliser ces étapes sont décrits dans les sections suivantes. Nous décrivons tout d'abord la façon dont l'ensemble des compositions **Cs** est généré. Puis nous indiquons la méthode de sélection des compositions valides. Nous établissons ensuite les critères de qualité permettant de trier les compositions et indiquons la méthode pour valider le résultat final des compositions. Nous définissons alors comment les résultats sont complétés.

Après ces sections théoriques, nous présentons un exemple qui développe le déroulement, étape par étape, de l'exécution du prototype. Nous terminons par une discussion sur ce prototype où nous abordons des cas particuliers que l'étape de validation sémantique ne peut pas résoudre, et nous analysons enfin le prototype dans son ensemble.

5.2 Génération des compositions par la fonction Fg

Comme nous l'avons dit précédemment, nous réutilisons ici la fonction de génération de composition Fg définie dans la section 4.2. Fondée sur la planification, elle utilise les types des données d'entrée et de sortie pour générer un graphe dont le parcours fournit une liste des compositions possibles.

5.3 Vérification de la validité des compositions (la fonction Fv)

Les méthodes basées sur la preuve de programme[11] peuvent être utilisées pour faire de la vérification de programme. La vérification consiste, par exemple, à comparer des spécifications avec des programmes et ainsi déterminer si ces programmes respectent les spécifications. Dans mon cas, je souhaite vérifier que les données réclamées par l'utilisateur seront bien celles fournies par la composition.

Les deux premières méthodes que nous avons vues dans ce chapitre, n'exploitent pas d'information sur les comportements des services Web mais uniquement des informations sur les types. Elles peuvent générer des compositions invalides. Pour limiter les erreurs, il est possible d'augmenter la granularité des types mais, ce faisant, nous augmentons la complexité du graphe en générant plus de noeuds et éliminons plus fréquemment des compositions qui auraient été valides.

La méthode basée sur Prolog va plus loin que les précédentes en exploitant des informations qui explicitent les transformations, réalisées par les services Web, sur les données.

Notre but est d'aller encore plus loin dans l'exploitation de la description des comportements des services Web en proposant un meilleur moyen de décrire ces comportements et une meilleure façon de comparer les compositions entre elles. Cette comparaison permettrait, par exemple, de déterminer si deux compositions sont interchangeables.

Nous définissons un langage de description du comportement des compositions de services Web. Ce langage s'inspire des formalisations des compositions des services Web présentées dans le chapitre 2 ainsi que des structures de contrôle utilisées dans OWL-S. La description du comportement des compositions passe par la connaissance du comportement de chacun des services Web participant à la composition et par la connaissance de la manière de combiner ces comportements.

5.3.1 Etablir le comportement d'un Web service

Pour connaître le comportement des services Web, on pourrait penser en première solution à utiliser les codes sources des services Web, de manière à relier finement les entrées aux sorties. Cette solution n'est pas réaliste. En effet, d'une part, personne ne fournit les codes sources des services Web et d'autre part, dans le cas où ces codes sources seraient disponibles, leur origine serait diverse et ils seraient codés dans des langages différents. Malgré la différence de langages, il reste théoriquement possible de traduire tous ces langages en un langage commun. Cette traduction, en dehors du temps très important qu'elle nécessiterait, se heurterait à d'autres problèmes comme, par exemple, la signification des types des données manipulées. Par exemple, une date peut se représenter de plusieurs manières différentes. De même, un nombre en virgule flottante peut représenter des données très différentes comme un rayonnement solaire, une température, un taux d'humidité, etc.

La solution consiste à définir un langage de description des comportements des services Web. Ce langage doit allier expressivité, simplicité d'utilisation et évolutivité. Il doit rester suffisamment abstrait de manière à ne pas prendre en compte des détails qui compliqueraient la description sans apporter de sens. Par exemple, il n'est pas utile de connaître la provenance géographique d'une donnée. Il doit être extensible de manière à prendre en compte de nouveaux types de données pour, par exemple, discriminer des températures de rayonnements solaires, ou des vitesses de vent. Trouver un juste niveau dans la description du comportement des services Web est difficile : il est impossible de trouver une limite objective entre un langage de type assembleur ne permettant plus de discriminer les types, et un langage de haut niveau comportant pléthore de types mais ne permettant plus de faire des équivalences de programmes.

Une composition de services peut être vue comme un service Web. En effet, si l'on reprend la formalisation de la composition de services Web, une composition est une façon de passer d'un environnement à un autre. En somme, si l'on peut définir un langage décrivant le comportement des services Web, ce langage devrait être également utilisé pour décrire le comportement des compositions. Par conséquent, je propose un langage ayant cette caractéristique.

Soulignons que ces choix de description sont subjectifs. Pour ma part, j'ai orienté mes choix en fonction des besoins exprimés par les météorologues. Ces besoins sont développés dans le chapitre 2 et ils m'imposent, par exemple, de pouvoir manipuler des séries temporelles.

5.3.2 Décomposition du langage

Le langage que je définis comporte des opérateurs permettant de décrire le comportement des services Web ainsi que les comportements des compositions. Dans cette section, je décris la façon dont ces opérateurs sont structurés ainsi que leur signification.

Une composition de services Web se décompose en quatre étapes :

1. la déclaration des services Web et de leur comportement ;
2. l'initialisation des données d'entrées ;
3. la description du comportement de la composition ;
4. l'exécution de la composition et le retour du résultat de la composition.

La composition *prog* est :

$$prog \quad := \mathbf{prgm}_p \; decl \; \mathbf{exec}_p \; c \; \mathbf{return}_p \; a$$

où : les caractères **gras** sont les mots clefs du langage ; *decl* est la partie déclaration des services Web et c décrit l'étape 3 et a décrit l'étape 4.

L'ensemble **DECLexp** des expressions de déclaration, l'ensemble **Cexp** des expressions de composition et l'ensemble **Aexp** des opérateurs arithmétiques sont définis dans la suite de cette section.

Notons que par la suite, par souci de simplicité, je confondrai les expressions et ce qu'elles représentent : l'expression d'une composition sera une composition, l'expression d'un entier sera un entier, etc. Dans certains cas, ce raccourci pourra ne pas être utilisé, par exemple, dans le cas des expressions arithmétiques.

5.3.3 Les types de données

5.3.3.1 Les types primitifs

Les spécifications de XML Schema [115] définissent les types primitifs suivants : *string*, *boolean*, *decimal*, *float*, *double*, *duration*, *dateTime*, *time*, *date*, *gYearMonth*, *gYear*, *gMonthDay*, *gDay*, *gMonth*, *hexBinary*, *base64Binary*, *anyURI*, *QName*, *NOTATION*. Les types primitifs sont des types atomiques, c'est-à-dire qu'ils sont indivisibles. Parmi ces types, j'ai choisi de n'intégrer à mon langage que les types qui m'ont paru essentiels pour réaliser un prototype. J'ai choisi de transcrire les types primitifs *string*, *boolean*, *decimal*, *float* en m'inspirant principalement de la syntaxe de JSON[49].

Un *string* est une chaîne de caractères. Chaque chaîne de caractères de l'ensemble **Sexp** est une suite de caractères alphanumériques, pouvant comporter des espaces entre deux ", par exemple : "*Hello World*".

Un *boolean* est une valeur qui représente un fait vrai ou faux. L'ensemble des **Bexp** est $\{\mathbf{true}, \mathbf{false}\}$. Les expressions **true** et **false** représentant respectivement vrai et faux.

Un *decimal* est un entier appartenant à l'ensemble $\mathbf{Z}$. Ces entiers sont représentés par l'ensemble **Zexp**, chaque expression z de **Zexp** est composée d'un groupe de chiffres, par exemple 999.

Un *float* est un nombre à virgule flottante appartenant à $\mathbf{R}$. Ces nombres sont représentés par l'ensemble **Rexp**, chaque expression r est construite par deux regroupements de chiffres séparés par un point, par exemple 99.9999.

Les autres types n'ont pas été retranscrits car ils peuvent être dérivés des types primitifs de mon langage. Par exemple, dateTime, gYearMonth, gYear, gMonthDay, gDay, gMonth sont des *decimal* et anyURI, QName sont des *string*.

5.3.3.2 Les types composés

La définition des types primitifs n'est pas suffisante pour représenter les données habituellement utilisées en informatique ou en météorologie. Les spécifications de XML Schema proposent plusieurs moyens de construire des types composés à partir de combinaisons de types primitifs, comme les unions et les types complexes. J'ai choisi de retranscrire ces moyens à l'aide de types particuliers. Ces types sont les listes et les dictionnaires, ces derniers étant connus sous le nom d'objet en JSON.

Les listes sont non typées et sont de longueur variable mais finie. Les listes correspondent à la cardinalité [0 ; *unbound*[des Schemas XML. L'ensemble **Lexp** des listes l est de la forme :

$$\begin{array}{rcl} l & := & \textbf{[}\ vlist\ \textbf{]} \mid \textbf{[ ]} \\ vlist & := & v \mid v\ \textbf{,}\ vlist \end{array}$$

où v est une valeur de **Vexp**. L'ensemble des valeurs **Vexp** est l'union des ensembles **Sexp**, **Zexp**, **Rexp**, **Lexp** et **Dexp**.

Les dictionnaires sont des listes de paires : identifiant, valeurs. Tout comme les listes, les dictionnaires peuvent être composés d'objets de type quelconque. L'ensemble des dictionnaires est **Dexp**. L'expression d'un dictionnaire d est construite récursivement par :

$$\begin{array}{rcl} d & := & \textbf{\{}\ pairlist\ \textbf{\}} \mid \textbf{\{ \}} \\ pairlist & := & id\ \textbf{:}\ v \mid pairlist\ \textbf{,}\ pairlist \end{array}$$

où id est un identifiant appartenant à **IDexp** et v est une valeur. L'ensemble des identifiants **IDexp** est composé de suites de caractères alphanumériques commençant par un caractère alphabétique.

Finalement, les données manipulées sont toujours des arbres, les types primitifs étant les feuilles de l'arbre, les dictionnaires et les listes formant des branches.

5.3.3.3 Notation pour la suite de cette section

- les variables s,s_0,s_1 et s_2 appartiennent implicitement à **Sexp**,
- les variables z,z_0,z_1 et z_2 appartiennent implicitement à **Zexp**,
- les variables r,r_0,r_1 et r_2 appartiennent implicitement à **Rexp**,
- les variables l,l_0,l_1 et l_2 appartiennent implicitement à **Lexp**,
- les variables d,d_0,d_1, d_2, Δ,Δ' et Δ'' appartiennent implicitement à **Dexp**,
- les variables v,v_0,v_1 et v_2 appartiennent implicitement à **Vexp**,
- les variables id,id_0,id_1 et id_2 appartiennent implicitement à **IDexp**.

5.3.4 Déclaration des services Web

J'ai choisi d'identifier un service Web à une fonction. Une fonction se définit par son entrée et sa sortie qui sont le plus souvent des dictionnaires et par une expression définissant son comportement. Je peux, de cette façon, définir un service Web fournissant une série temporelle de température, à l'aide d'un service Web fournissant une

seule valeur de température en un point spatio-temporel donné, comme le montrera l'exemple de ce chapitre. Les fonctions peuvent faire appel à d'autres services Web représentés aussi sous forme de fonctions. Ce type d'appel est limité en interdisant les appels récursifs (une fonction qui s'appelle elle-même) de manière à garantir la terminaison de l'exécution. En pratique, lorsque une fonction est appelée pour la première fois, on la retire de la liste des fonctions éligibles, puis, une fois l'exécution de cette fonction terminée, elle est remise dans la liste des fonctions éligibles. De cette façon, cette fonction devient inexistante pour toutes les sous-fonctions utilisées par la fonction courante.

Certains services Web ne peuvent pas être définis de cette façon. Par exemple, le service Web fournissant des températures en un point spatio-temporel (que j'utiliserai dans l'exemple de ce chapitre) est difficile à définir à partir d'autres services Web. Pour cette raison, il est utile de pouvoir définir des fonctions dont le comportement est inconnu qui représentent ces services Web. En météorologie, ces services, que j'appelle *services de base*, sont généralement des services Web fournissant une unique valeur physique. Ces services Web sont définis par des fonctions copiant les données d'entrée représentant les données qu'ils devraient fournir dans un dictionnaire en y ajoutant l'identifiant $final$ associé à la chaîne de caractères représentant le type de la donnée. La caractéristique de ces services Web de base est qu'ils ne peuvent pas être comparés à d'autres services Web, étant donné que leur comportement et leurs propriétés sont inconnus. Ils sont donc seulement égaux à eux-mêmes.

Les fonctions sont déclarées sous forme de listes de la façon suivante :

$$decl \quad := \quad \textbf{func}\ id\ \textbf{exec}\ c_1\ \textbf{return}\ a_1\ decl_0 |\ \textbf{none}$$

Les fonctions sont déclarées à l'aide des mots clefs **func** ... **exec** ... **return**.... Elles sont définies par un identifiant id, une expression de composition c et une expression arithmétique a. Dans le cas d'une fonction de *base*, le rôle de l'expression c est de recopier les entrées en y ajoutant l'identifiant clef $final$ avec la chaîne de caractères définissant le type de la donnée représentée. Par exemple, pour le service Web fournissant une température en un point spatio-temporel, l'expression recopiera les valeurs de latitude, de longitude et de date, et ajoutera $final$ avec le type "*temperature*". De cette façon, il sera possible de comparer deux expressions faisant appel à une fonction de base en utilisant les données enregistrées dans cette valeur spéciale.

5.3.5 Les expressions sur les données

Ces expressions sont destinées à effectuer des calculs sur les données et à créer de nouvelles valeurs. L'ensemble **Aexp** est l'ensemble des opérations a telles que :

a := b | z | r | s | l | d |
id (a_0) | id | a_0 . id | **this** |
range (a_0, a_1) |
append a_0 a_1 |
$a_0 + a_1$ | a_0 - a_1 | a_0 / a_1 | a_0 * a_1 | a_0 % a_1 |
a_0 **or** a_1 | a_0 **and** a_1 | **not** a_0 | $a_0 > a_1$ | $a_0 == a_1$ | **str** (a_0)

où :

- les expressions b, z, r, s, l et d créent de nouvelles données dans l'environnement ;

- l'opérateur $id(a_0)$ appelle les fonctions déclarées dans la partie déclaration du programme.
- le langage fournit trois manières d'accéder aux données qui seront spécifiées dans la section suivante :
  - l'expression id permet d'obtenir la valeur identifiée par id dans l'environnement local ;
  - l'expression $a_0.id$ renvoie la valeur identifiée par id dans le dictionnaire renvoyé par a_0 ;
  - l'opérateur **this** renvoie comme valeur l'environnement complet.
- l'opérateur **range**(a_0, a_1) génère des listes d'entiers entre les deux paramètres arguments a_0 et a_1 ;
- la fonction **append l e** permet d'ajouter un nouvel élément e en fin d'une liste l ;
- les opérateurs arithmétiques classiques +, -, *, /, <, == manipulent des entiers ou des nombres à virgule flottante ;
- les opérateurs booléens classiques **or**, **and**, **not** manipulent les booléens ;
- l'opérateur **str** transforme un entier, un booléen ou un nombre réel en une chaîne de caractères ;

La section sur la sémantique opérationnelle décrit en détail les comportements de ces opérateurs.

5.3.6 Les opérateurs de contrôle

La description du comportement des compositions est également utilisée pour la définition des types et des fonctions. L'ensemble des compositions **Cexp** est l'ensemble des expressions c de la forme suivante :

$$c \ := c_0 \ ; c_1 \mid p = a \mid \textbf{skip} \mid \textbf{if } a \textbf{ then } c_0 \textbf{ else } c_1 \textbf{ end} \mid \textbf{foreach } p \textbf{ in } a \textbf{ do } c \textbf{ done}$$

où p représente un chemin dans l'environnement. L'ensemble des chemins de l'environnement **Pexp** est l'ensemble des expressions p telles que :

$$p \quad := \quad id \mid id \ . \ p_0$$

Dans le groupe des opérateurs de contrôle :

- l'opérateur classique des langages impératifs " **;**" permet d'enchaîner des opérations en séquence ;
- l'opérateur d'affectation $p = a$ associe au chemin p la valeur renvoyée par a ;
- l'opérateur **skip** ne fait rien. Il est utile pour les opérateurs **if ... then ... else ... end** en particulier.
- l'opérateur de test **if ... then ... else ... end** exécute la première expression si le résultat du test est **true** ou la seconde si le test est égal à **false** ;
- l'opérateur **foreach ... in ... do ... done** s'applique aux listes en permettant de les parcourir ;
- le chemin p décrit un chemin dans les dictionnaires de l'environnement ;

5.3.7 La sémantique opérationnelle du langage

Maintenant que nous avons une description informelle du comportement des services Web et un aperçu des comportements des opérateurs du langage, nous formalisons dans cette section ces comportements à l'aide d'une sémantique opérationnelle.

Comme nous l'avons déjà vu dans le chapitre dédié à la formalisation de la composition de services Web, établir notre sémantique opérationnelle va consister tout d'abord à définir l'environnement sur lequel cette sémantique travaille, ensuite à définir les règles opérationnelles du comportement des opérateurs du langage, en commençant par les opérateurs manipulant l'environnement et en terminant sur les opérateurs liés au contrôle du déroulement du programme.

5.3.7.1 Notation pour la suite de cette section

- les variables a,a_0,a_1 et a_2 appartiennent implicitement à **Aexp** ;
- les variables c,c_0,c_1 et c_2 appartiennent implicitement à **Cexp** ;
- les variables p,p_0,p_1 et p_2 appartiennent implicitement à **Pexp**.

5.3.7.2 L'environnement d'exécution des services Web

L'environnement correspond à l'environnement d'exécution de la composition. Un ordinateur peut être vu comme une machine à états qui, après chaque instruction, change d'état. Dans mon langage, l'environnement est représenté par une paire (Δ, Φ), où Δ est un dictionnaire représentant les données de l'état courant et Φ est une fonction donnant accès à la définition des fonctions. Δ appartient à **Dexp**.

5.3.7.3 Accès aux données de l'environnement

Pour accéder aux données de l'environnement, on dispose des deux fonctions suivantes :

- la fonction $d(id)$ qui renvoie la valeur v associée à l'identifiant id dans le dictionnaire d ;
- la fonction $d[id \mapsto v]$:

$$d[id \mapsto v](i) = \begin{cases} d(i) & \text{si } i \neq id \\ v & \text{si } i = id \end{cases} \tag{5.1}$$

qui renvoie v si id et i coïncident et $d(i)$ sinon. Notons que cette fonction définit un *nouveau* dictionnaire identique au précédent mais dans lequel id est associé à v. Cette opération peut être vue comme l'ajout d'une entrée dans le dictionnaire d.

5.3.7.4 Accès aux fonctions de l'environnement

L'environnement comporte une fonction Φ qui associe à un identificateur de fonction id une paire (c, a) où c représente le comportement de la fonction id et a la valeur de retour de la fonction id.

Φ donne accès aux définitions des fonctions représentant :

- les services Web réels ;
- les fonctions représentant des services Web virtuels ;
- les fonctions représentant des services dont le comportement est inconnu.

De même que pour la fonction Δ, je définis les opérateurs :
- $\Phi(id)$ qui fournit la paire (c, a) correspondant à id dans Φ ;
- $\Phi[id \mapsto (c, a)]$. $\Phi(id)$ qui représente la fonction :

$$\Phi[id \mapsto (c,a)](i) = \begin{cases} \Phi(i) & \text{si } i \neq id \\ (c_{expr}, a_{expr}) & \text{si } i = id \end{cases} \tag{5.2}$$

Notons que les listes sont définies à l'aide des opérateurs suivants :
- l'opérateur $v :: l$ ajoute en tête de la liste l l'élément v ;
- l'opérateur $head(l)$ accède à l'élément en tête de la liste l ;
- l'opérateur $tail(l)$ accède au reste de la liste une fois le premier élément enlevé.

5.3.7.5 Evaluation d'une expression dans l'environnement

La règle $\Delta, \Phi \vdash e \rightarrow_a v$ signifie que dans l'environnement Δ, Φ, l'expression e s'évalue de façon arithmétique $\rightarrow_a$ en la valeur v.

5.3.7.6 Evaluation des expressions sur les types de base

Les expressions z, r, s, l, d permettent de créer les types de base. Ils sont définis de la façon suivante :

$$\frac{}{\Delta, \Phi \vdash z \rightarrow_a z} \tag{5.3}$$

$$\frac{}{\Delta, \Phi \vdash r \rightarrow_a r} \tag{5.4}$$

$$\frac{}{\Delta, \Phi \vdash s \rightarrow_a s} \tag{5.5}$$

$$\frac{}{\Delta, \Phi \vdash l \rightarrow_a l} \tag{5.6}$$

$$\frac{}{\Delta, \Phi \vdash d \rightarrow_a d} \tag{5.7}$$

Notons que pour simplifier la définition des opérateurs arithmétiques, je confonds l'expression à évaluer avec sa valeur d'évaluation. Ainsi dans $\Delta, \Phi \vdash z \rightarrow_a z\ z$ c'est le même identificateur z que nous retrouvons des deux côtés de $\rightarrow_a$.

5.3.7.7 L'opérateur *range*

Une autre façon de créer des listes est l'utilisation de l'opérateur **range**. Dans mon langage, il est limité à la création de liste d'entiers et se définit de la façon suivante :

$$\frac{\Delta_0, \Phi \vdash a_0 \rightarrow_a z_1 \quad \Delta_0, \Phi \vdash a_1 \rightarrow_a z_2}{\Delta_0, \Phi \vdash \textbf{range}(a_0, a_1) \rightarrow_a l} \tag{5.8}$$

où l est l'expression de la liste des entiers entre z_1 et z_2. Notons que nous restreignons l'opérateur **range** à n'utiliser que des bornes entières.

5.3.7.8 Les opérateurs arithmétiques

Les opérateurs $+$, $-$, $*$, $/$ s'appliquent aux entiers **Zexp** et aux réels **Rexp**, les définitions de l'opérateur $+$ pour les entiers et les réels sont données par les deux règles ci-dessous :

$$\frac{\Delta, \Phi \vdash a_0 \rightarrow_a z_0 \quad z_0 \in Zexp \quad \Delta, \Phi \vdash a_1 \rightarrow_a z_1 \quad z_1 \in Zexp}{\Delta, \Phi \vdash a_0 + a_1 \rightarrow_a z_0 + z_1} \tag{5.9}$$

$$\frac{\Delta, \Phi \vdash a_0 \rightarrow_a r_0 \quad r_0 \in Rexp \quad \Delta, \Phi \vdash a_1 \rightarrow_a r_1 \quad r_1 \in Rexp}{\Delta, \Phi \vdash a_0 + a_1 \rightarrow_a r_0 + r_1} \tag{5.10}$$

Notons que l'opérateur + situé à droite des $\rightarrow_a$ est dans la première règle l'opérateur + des entiers et dans la seconde, l'opérateur + des réels.

Sur le même modèle, nous pouvons exprimer les opérateurs $-$, $*$ et $/$:

$$\frac{\Delta, \Phi \vdash a_0 \rightarrow_a z_0 \quad z_0 \in Zexp \quad \Delta, \Phi \vdash a_1 \rightarrow_a z_1 \quad z_1 \in Zexp}{\Delta, \Phi \vdash a_0 - a_1 \rightarrow_a z_0 - z_1} \tag{5.11}$$

$$\frac{\Delta, \Phi \vdash a_0 \rightarrow_a r_0 \quad r_0 \in Rexp \quad \Delta, \Phi \vdash a_1 \rightarrow_a r_1 \quad r_1 \in Rexp}{\Delta, \Phi \vdash a_0 - a_1 \rightarrow_a r_0 - r_1} \tag{5.12}$$

$$\frac{\Delta, \Phi \vdash a_0 \rightarrow_a z_0 \quad z_0 \in Zexp \quad \Delta, \Phi \vdash a_1 \rightarrow_a z_1 \quad z_1 \in Zexp}{\Delta, \Phi \vdash a_0 * a_1 \rightarrow_a z_0 * z_1} \tag{5.13}$$

$$\frac{\Delta, \Phi \vdash a_0 \rightarrow_a r_0 \quad r_0 \in Rexp \quad \Delta, \Phi \vdash a_1 \rightarrow_a r_1 \quad r_1 \in Rexp}{\Delta, \Phi \vdash a_0 * a_1 \rightarrow_a r_0 * r_1} \tag{5.14}$$

$$\frac{\Delta, \Phi \vdash a_0 \rightarrow_a z_0 \quad z_0 \in Zexp \quad \Delta, \Phi \vdash a_1 \rightarrow_a z_1 \quad z_1 \in Zexp}{\Delta, \Phi \vdash a_0/a_1 \rightarrow_a z_0/z_1} \tag{5.15}$$

$$\frac{\Delta, \Phi \vdash a_0 \rightarrow_a r_0 \quad r_0 \in Rexp \quad \Delta, \Phi \vdash a_1 \rightarrow_a r_1 \quad r_1 \in Rexp}{\Delta, \Phi \vdash a_0/a_1 \rightarrow_a r_0/r_1} \tag{5.16}$$

5.3.7.9 L'opérateur de manipulation d'une liste

L'opérateur **append** permet de rajouter une valeur à la fin d'une liste, et s'exprime de la façon suivante :

$$\frac{\Delta, \Phi \vdash a_1 \rightarrow_a v_1 \; \Delta, \Phi \vdash a_0 \rightarrow_a l_0}{\Delta, \Phi \vdash \textbf{append} \; a_0 \; a_1 \rightarrow_a v_1 :: l_0} \tag{5.17}$$

5.3.7.10 L'accès aux données d'un dictionnaire

L'accès aux données d'un dictionnaire est exprimé par les 3 règles suivantes.

La première règle exprime l'opérateur qui renvoie tout l'environnement :

$$\frac{}{\Delta, \Phi \vdash \textbf{this} \rightarrow_a \Delta} \tag{5.18}$$

La seconde règle exprime l'opérateur qui s'évalue en la valeur associée à id dans l'environnement :

$$\frac{}{\Delta, \Phi \vdash id \rightarrow_a \Delta(id)} \tag{5.19}$$

Enfin, la dernière règle fournit la valeur associée à id dans le dictionnaire d que doit fournir l'expression a_0 :

$$\frac{\Delta, \Phi \vdash a_0 \rightarrow_a d}{\Delta, \Phi \vdash a_0 \; . \; id \rightarrow_a d(id)} \tag{5.20}$$

En combinant ces trois opérateurs nous pouvons ainsi accéder à toutes les données de chaque dictionnaire, par exemple l'expression : **this**.$a.b$ renvoie la valeur associée à b dans le dictionnaire de l'environnement local associé à a.

5.3.7.11 L'opérateur d'appel de fonction

L'opérateur d'appel de fonction est considéré comme une opération arithmétique, car il ne modifie pas l'environnement local.

Comme je l'ai décrit dans la section 5.3.4, il existe deux catégories de fonctions : les fonctions de bases et les autres. Le langage ne différencie pas ces deux catégories. Les fonctions de base sont des fonctions classiques qui suivent une convention particulière, un exemple dans la section 5.7 en fin de ce chapitre l'illustre.

Par ailleurs, l'appel des fonctions est défini par :

$$\frac{\begin{array}{c}\Delta, \Phi \vdash a_0 \rightarrow_a \Delta' \quad \Phi(id) = (c, a_1) \\ \Delta', \Phi[id \mapsto (\textbf{skip}, \textbf{this})] \vdash c \rightarrow_c \Delta'' \\ \Delta'', \Phi[id \mapsto (\textbf{skip}, \textbf{this})] \vdash a_1 \rightarrow_a v\end{array}}{\Delta, \Phi \vdash id(a_0) \rightarrow_a v} \tag{5.21}$$

où :

- Δ' est l'environnement spécifique dans lequel travaillent les fonctions. Cet environnement est indépendant de l'environnement d'exécution. On exprime de cette manière que les comportements des services Web n'ont accès qu'aux données fournies en entrée du service Web ;
- a_0 est l'expression arithmétique chargée de fournir les données à la fonction qui va être exécutée. Généralement, le résultat de l'évaluation de a_0 est un dictionnaire ;
- c est l'expression de la fonction. Cette expression est évaluée à l'aide de la notation $\rightarrow_c$ définie plus loin ;
- a_1 est l'expression arithmétique de la valeur de retour de la fonction à évaluer. Cette expression est évaluée après l'évaluation de l'expression c. L'évaluation de ces expressions est réalisée en ayant au préalable supprimé de l'environnement la fonction en cours d'appel, afin d'éviter les appels récursifs.

5.3.7.12 Les opérateurs de contrôle

La manipulation de l'environnement s'effectue par l'intermédiaire de l'évaluation des opérateurs de contrôle qui permettent d'exprimer les compositions.

$\Delta, \Phi \vdash a \rightarrow_c v$ signifie que dans l'environnement Δ, Φ, l'expression a s'évalue dans l'environnement en v.

L'opérateur skip Le premier opérateur de contrôle est **skip**. Cet opérateur ne fait rien, c'est-à-dire qu'il ne change pas l'environnement. Il est défini de la façon suivante :

$$\frac{}{\Delta, \Phi \vdash \textbf{skip} \rightarrow_c \Delta} \tag{5.22}$$

Cet opérateur est utile pour la définition des opérateurs de test en permettant d'exprimer que les expressions des branches **then** ou **else** sont vides, c'est-à-dire n'ont pas de comportement.

L'opérateur de séquence ; C'est l'un des opérateurs les plus utilisés. Il consiste à exécuter l'une après l'autre les deux expressions, la première transformant l'environnement dans lequel sera être exécutée la seconde. Il est défini par :

$$\frac{\Delta, \Phi \vdash c_0 \rightarrow_c \Delta' \quad \Delta', \Phi \vdash c_1 \rightarrow_c \Delta''}{\Delta, \Phi \vdash c_0 \ ; c_1 \rightarrow_c \Delta''} \tag{5.23}$$

où c_0 et c_1 sont des expressions de composition du langage.

L'opérateur d'affectation Cet opérateur permet d'associer un identifiant à une valeur. L'affectation modifie récursivement les dictionnaires de l'environnement de la manière suivante :

- d'abord elle modifie le dictionnaire le plus profond dans l'arborescence ;
- puis elle modifie récursivement les dictionnaires de niveau immédiatement inférieur jusqu'à atteindre le dictionnaire racine.

Cet opérateur se décompose dans les trois règles récursives explicitées ci-dessous.

La première règle 5.24 exprime le fait que l'on modifie le dictionnaire de niveau inférieur avec le dictionnaire de niveau immédiatement supérieur, Δ étant le dictionnaire de niveau inférieur et $\Delta(id)$ le dictionnaire de niveau immédiatement supérieur. $\Delta(id)$ est modifié pour donner Δ', puis ce $\Delta(id)$ modifié est réaffecté à id. Notons que Δ' est plus loin de la racine que Δ et dépend de $\Delta(id)$.

$$\frac{\Delta(id), \Phi \vdash p =_{rec} v_1 \rightarrow_c \Delta'}{\Delta, \Phi \vdash id \ . \ p =_{rec} v_1 \rightarrow \Delta[id \mapsto \Delta']} \tag{5.24}$$

La seconde règle 5.25 exprime le fait que l'on a atteint le dictionnaire le plus éloigné de la racine. La valeur fournie par l'expression a_{expr} de la règle suivante est donc ajoutée au dictionnaire qui va être propagé.

$$\frac{}{\Delta, \Phi \vdash id =_{rec} v_1 \rightarrow_c \Delta[id \mapsto v_1]} \tag{5.25}$$

C'est avec cette dernière règle que débute la récursion. Cette règle évalue l'expression a_{expr} puis lance la récursion. Le chemin p représente l'endroit où doit être enregistrée la valeur. Par exemple, le chemin $a.b.c$ indique la valeur c du dictionnaire b, qui est lui-même dans le dictionnaire a de l'environnement local. Modifier la valeur v de c revient à modifier tous les dictionnaires a, b ainsi que le dictionnaire local. La formule est : $\Delta' = \Delta[a \mapsto \Delta(a)[b \mapsto \Delta(a)(b)[c \mapsto v]]]$. Une fois l'affectation achevée, l'évaluation arithmétique de $a.b.c$ sera égale à v

$$\frac{\Delta, \Phi \vdash a_{expr} \rightarrow_a v_1 \quad \Delta, \Phi \vdash p =_{rec} v_1 \rightarrow_c \Delta'}{\Delta, \Phi \vdash p = a_{expr} \rightarrow_a \Delta'} \tag{5.26}$$

L'opérateur de branchement conditionnel *if ... then ... else ... end* Cet opérateur calcule le résultat d'une expression booléenne pour déterminer quelle branche de composition sera exécutée. Il y a donc deux cas possibles :

$$\frac{\Delta, \Phi \vdash a_0 \rightarrow_a \textbf{false} \quad \Delta, \Phi \vdash c_2 \rightarrow_c \Delta'}{\Delta, \Phi \vdash \textbf{if } a_0 \textbf{ then } c_1 \textbf{ else } c_2 \textbf{ end} \rightarrow_c \Delta'} \tag{5.27}$$

$$\frac{\Delta, \Phi \vdash a_0 \rightarrow_a \textbf{true} \quad \Delta, \Phi \vdash c_1 \rightarrow_c \Delta'}{\Delta, \Phi \vdash \textbf{if } a_0 \textbf{ then } c_1 \textbf{ else } c_2 \textbf{ end} \rightarrow_c \Delta'} \tag{5.28}$$

La boucle *foreach ... in ... do ... done* L'opérateur de boucle utilisé par le langage se définit en deux étapes :

1. la première étape extrait la liste de la boucle **foreach**, c'est le rôle de l'opérateur **foreach** ;
2. la suivante copie et parcourt cette liste, c'est le rôle de l'opérateur $\textbf{foreach}_{rec}$.

Dans cette première règle, j'initialise le **foreach** en évaluant a pour obtenir la liste correspondante qui sera ensuite parcourue grâce aux deux règles suivantes.

$$\frac{\Delta, \Phi \vdash a \rightarrow_a l \quad \Delta, \Phi \vdash \textbf{foreach}_{rec}\ p\ \textbf{in}_{rec}\ l\ \textbf{do}_{rec}\ c\ \textbf{done} \rightarrow_c \Delta'}{\Delta, \Phi \vdash \textbf{foreach}\ p\ \textbf{in}\ a\ \textbf{do}\ c\ \textbf{done} \rightarrow_c \Delta'} \tag{5.29}$$

La seconde règle définit le fait que : faire un **foreach** sur la liste l revient à affecter la première valeur de la liste à p puis à évaluer c et enfin à évaluer un **foreach** sur le reste de la liste.

$$\frac{\Delta, \Phi \vdash p =_{rec} head(l)\ ;\ c\ ;\ \textbf{foreach}_{rec}\ p\ \textbf{in}_{rec}\ rest(l)\ \textbf{do}_{rec}\ c\ \textbf{done} \rightarrow_c \Delta'}{\Delta, \Phi \vdash \textbf{foreach}_{rec}\ p\ \textbf{in}_{rec}\ l\ \textbf{do}_{rec}\ c\ \textbf{done} \rightarrow_c \Delta'} \tag{5.30}$$

Dans cette dernière règle, **[ ]** représente la liste vide qui signifie la fin du **foreach**.

$$\frac{}{\Delta, \Phi \vdash \textbf{foreach}_{rec}\ p\ \textbf{in}_{rec}\ \textbf{[ ]}\ \textbf{do}_{rec}\ c\ \textbf{done} \rightarrow_c \Delta} \tag{5.31}$$

L'opérateur de déclaration des services La déclaration des services modifie la fonction Φ de l'environnement. Pour exprimer cette règle, je définis tout d'abord la notation $\Phi \vdash decl \rightarrow_f \Phi'$ pour exprimer que la déclaration $decl$ dans l'environnement Φ modifie cet environnement en Φ'.

La règle exprime la fin des déclarations des fonctions, ou plus généralement aucune déclaration de fonction.

$$\frac{}{\Phi \vdash \textbf{none} \rightarrow_f \Phi} \tag{5.32}$$

La seconde règle exprime deux choses :

1 le remplacement ou l'affectation à id d'une fonction dont le corps et la valeur sont représentés par la paire (c, a) ;

2 la séquence des déclarations qui sont réalisées à la suite de la déclaration par l'expression $decl$.

La déclaration des fonctions est globale. Ces fonctions ne pourront pas être modifiées durant l'exécution des expressions **Cexp** ou **Aexp**. Seule la dernière déclaration d'une fonction est effective.

$$\frac{\Phi[id \mapsto (c, a)] \vdash decl \rightarrow_f \Phi'}{\Phi \vdash \textbf{func}\ id\ \textbf{exec}\ c\ \textbf{return}\ a\ decl\ \rightarrow_f \Phi'} \tag{5.33}$$

5.3.8 Exécution théorique d'une composition

Notre but ici est de réaliser F_v qui compare les différentes compositions possibles avec la composition souhaitée par l'utilisateur. Nous commençons tout d'abord par déterminer les compositions pouvant répondre à la demande de l'utilisateur. Chaque séquence obtenue est ensuite traduite dans une expression **PROGexp** du langage décrit précédemment, cette expression représentant la composition. Toutes les expressions sont ensuite évaluées de manière à obtenir une valeur v représentant leur résultat. Cette évaluation est décrite par la règle suivante :

$$\frac{\Phi^0 \vdash decl \rightarrow_f \Phi \quad \{\ \}, \Phi \vdash c \rightarrow_c \Delta \quad \Delta, \Phi \vdash a \rightarrow_a v}{\vdash \textbf{prog}\ decl\ \textbf{exec}\ c\ \textbf{return}\ a\ \rightarrow_p v} \tag{5.34}$$

où :
Φ^0 est la fonction pour laquelle aucun identifiant n'est associé à une fonction ;
Φ est la fonction de l'environnement donnant accès aux fonctions déclarées par l'expression *decl* ;
c est l'expression de la composition qui va être exécutée ;
a est l'expression arithmétique qui donnera la valeur v de retour de l'évaluation de la composition.

L'exécution de la composition conduit à la valeur finale v correspondant au résultat de la composition. Cette valeur peut être de type primitif. Elle peut aussi être une combinaison de données de types complexes représentant l'exécution de la composition réelle ; nous en verrons un exemple typique dans la section 5.7. Ce résultat peut être comparé à un autre résultat, de la façon décrite dans la section suivante, afin de déterminer si les compositions sont équivalentes.

5.3.9 Validation théorique des compositions trouvées

Les résultats de l'interprétation des compositions sont comparés en étudiant récursivement les dictionnaires, les listes et les valeurs de sortie qui ont été générés. Cette validation théorique correspond à la phase ii : *il valide ces compositions à l'aide d'un processus de vérification qui utilise la description sémantique des services Web et de la requête de l'utilisateur* (cf. §5.1). Si les compositions ne sont pas validées elles sont éliminées (phase iii).

On vérifie si toutes les données souhaitées par l'utilisateur sont obtenues en sortie. Pour ce faire, j'ai défini une fonction *subseq* dont le but est de déterminer si un ensemble a est un sous-ensemble de b. Cette fonction retourne True si a est un sous-ensemble de b et False sinon. L'algorithme de cette fonction se base sur le type de a et de b pour déterminer comment la comparaison doit être réalisée. Il se déroule de la manière suivante :

- quand les types de a et b diffèrent, alors a ne peut pas être un sous-ensemble de b et la fonction *subseq* renvoie False ;

- quand a et b sont de même type :
  - si a et b appartiennent à l'un des ensembles suivants **Zexp**, **Rexp**, **Sexp** ou **Bexp**, alors a est un sous-ensemble de b si a est égal à b ;
  - si a et b sont des dictionnaires, alors a est un sous-ensemble de b si et seulement si pour toutes les paires $(id_a, v_a) \in a$, il existe une paire $(id_b, v_b) \in b$ telle que id_a égale id_b et $subseq(v_a, v_b)$ égale à True ;
  - si a et b sont des listes, alors a est un sous-ensemble de b si et seulement si pour toute valeur $v_a \in a$, il existe une valeur $v_b \in b$ telle que $subseq(v_a, v_b)$ est True. L'algorithme utilisé considère les listes comme des ensembles de valeurs non ordonnés.

Je présente l'algorithme écrit dans le langage Python, qui comporte les mêmes types de base que ceux utilisés par la méthode de description des environnements. Cette méthode comporte également l'opérateur "type" permettant d'obtenir le type d'une valeur.

```
def subseq (a, b) :
    if type(a) != type(b) :
        return False
    if type(b) != list and type(b) != dict :
        return b == a
    elif type(b) == dict :
        for k in a.keys() :
            if b.has_key(k) :
                if not subseq(a[k], b[k]) :
                    return False
            else :
                return False
            return True
    elif type(b) == list :
        for i in a :
            issubsub = False
            for j in b :
                if subseq (i, j) :
                    issubsub = True
            if issubsub == False :
                return False
        return True
    return False
```

On remarque que la comparaison n'est pas commutative, c'est-à-dire que si $prog_1$ peut remplacer $prog_2$, cela n'implique pas que $prog_2$ peut remplacer $prog_1$. Par exemple, si la composition $prog_1$ permet d'obtenir une quantité de rayonnement infra-rouge et une quantité de rayonnement ultra-violet et qu'une composition $prog_2$ permet d'obtenir une quantité de rayonnement ultra-violet, alors $prog_1$ peut remplacer $prog_2$ parce que toutes les données fournies par $prog_2$ sont aussi fournies par $prog_1$, mais $prog_2$ ne peut pas remplacer $prog_1$ pour la raison inverse.

5.4 Prise en compte de la qualité

Dans cette phase iv du prototype hybride : *il trie les compositions en fonction de leur qualité présumée*, la qualité peut être prise en compte de plusieurs manières : la première consiste à définir la qualité des services Web et la seconde consiste à définir la qualité des données finales.

5.4.1 Prise en compte de la qualité des services Web

La prise en compte de la qualité des services Web peut se faire de la même façon que pour la méthode de composition basée sur les graphes : en calculant la qualité de chaque composition en fonction des qualités des services Web qui la composent. Dans mon prototype, j'ai adopté cette approche.

5.4.2 Prise en compte de la qualité des données finales

Cette méthode permet de prendre en compte la qualité au niveau de la donnée ou, plus exactement, au niveau des fonctions de base. Lorsqu'un service Web est défini à partir d'une fonction de base, il rajoute une donnée de qualité au résultat de l'appel de la fonction de base. Ces fonctions de base représentant des données de base, il est alors possible d'obtenir une évaluation de la qualité donnée par donnée. De plus, chaque service Web pourra modifier les données de qualité de façon plus fine.

Par exemple, la définition d'une fonction de getIrradUVA de la manière suivante :

```
func getIrradUVA exec
    tmp = { final : "irradUVA" } ;
    tmp.date = date;
    tmp.geopoint = geopoint;
    tmp.quality = quality
    tmp.unit = unit;
return tmp
```

et le service Web définirait la qualité de ce service Web grâce à :

```
func ServiceA exec
    result = [ ];
    foreach i in range(period.begin, period.end) do
        tmp = { sequence : [ ] };
        tmp.date = i;
        tmp.geopoint = geopoint;
        tmp.unit = "JM2";
        tmp.quality = 0.5
        result.sequence = append(result.sequence, getIrradUVA(tmp))
    done
return result
```

La valeur 0.5 correspond à une valeur de qualité pour une donnée. Dans cet exemple, toutes les données du serviceA ont une qualité de 0.5, néanmoins il est possible d'écrire une description qui définirait, par exemple, une qualité de 0.9 pour le mois de janvier et une autre valeur pour les autres mois. Ainsi, chaque service Web peut définir la qualité de chacune des données. Je n'ai pas réalisé cette méthode, mais si je l'avais fait il aurait été également nécessaire de changer la méthode de tri des méthodes de composition, d'union et de validation.

5.5 Union automatique des résultats de l'exécution des compositions

Après avoir exécuté les compositions, le résultat doit être combiné avec les résultats précédemment obtenus (phase v). Il est possible de réaliser cette union de résultats de façon automatique compte-tenu de la façon dont les données sont enregistrées. L'union des données a et b est réalisée de manière à compléter les dictionnaires et les listes a avec b.

Les dictionnaires sont complétés récursivement en comparant les entrées présentes dans le dictionnaire b et manquantes dans a et, dans ce cas, une entrée est alors créée dans le dictionnaire a, avec la même valeur que celle contenue dans le dictionnaire b.

Dans le cas où les entrées sont présentes dans les deux dictionnaires, alors l'entrée présente dans a, est complétée par l'entrée qui est dans b.

Comme pour la comparaison, les listes sont traitées comme des ensembles. Les listes sont complétées en recherchant pour toutes les valeurs de la liste b une valeur qui soit un sous-ensemble de cette valeur, à l'aide de la fonction *subseq* définie ci-dessus. Lorsqu'une correspondance est trouvée, alors la valeur contenue dans a est complétée par la valeur contenue dans b. Si aucune correspondance n'est trouvée, alors cette valeur est ajoutée à la liste a.

L'algorithme d'union des résultats a et b est le suivant :

```
def union (a, b) :
    if type(a) != type(b) :
        return None
    if type(b) == dict :
        for k in b.keys() :
            if a.has_key(k) :
                union(a[k], b[k])
            else :
                a[k] = b[k]
    elif type(b) == list :
        for i in b :
            issubsub = False
            for j in a :
                if subseq (i, j) :
                    issubsub = True
                    break
            if issubsub == False :
                a.append(i)
```

```
        return True
    return False
```

5.6 Validation du résultat de l'exécution réelle des compositions

Une fois l'union des résultats réels réalisée, il faut vérifier simplement que toutes les données souhaitées par l'utilisateur sont présentes dans le résultat réel (phase vi). Pour valider le résultat réel, le prototype applique la fonction *validate*. Cette fonction utilise en entrée :

- le résultat a de l'interprétation de la requête de l'utilisateur ;
- et le résultat b de l'union des exécutions réelles des compositions de services Web.

Le résultat a représente le résultat que souhaite obtenir l'utilisateur, et b représente le résultat réel que la méthode de composition est parvenue à obtenir. Le résultat est donc validé si et seulement si le résultat a est un sous-ensemble du résultat b, en ignorant le mot clef "final". En effet, ce mot clef est présent dans le résultat théorique a de la requête de l'utilisateur mais il ne l'est pas dans l'union des résultats réels. La vérification s'effectue de manière similaire à la validation des compositions :

```
def validate (a, b) :
    if type(a) != type(b) :
        return False
    if type(b) != list and type(b) != dict :
        return b == a
    elif type(b) == dict :
        for k in a.keys() :
            if k == 'final' :
                pass
            elif b.has_key(k) :
                if not validate(a[k], b[k]) :
                    return False
            else :
                return False
        return True
    elif type(b) == list :
        for i in a :
            issubsub = False
            for j in b :
                if validate(i, j) :
                    issubsub = True
            if issubsub == False :
                return False
        return True
    return False
```

Le paramètre a est le résultat de la requête de l'utilisateur, et le paramètre b le résultat de l'exécution de la composition et de son union avec les résultats précédents. La fonction *validate* renvoie $True$ si la composition est validée et renvoie $False$ si la composition est refusée. Cette fonction est la même que la fonction *subseq* définie dans la section 5.3.9, à la seule différence que cette fonction ignore l'identifiant clef "final".

5.7 Exemple d'application : l'utilisateur de Lyon

Je présente dans cette section, un exemple concret d'application de la méthode proposée. Cette application illustre le fonctionnement de la méthode et introduit une discussion sur les qualités et les limitations de cette méthode.

Pour commencer, je propose une façon compacte d'écrire les programmes de manière à les rendre plus lisibles. Ensuite, je décris les services Web nécessaires à l'illustration. Ces services Web ont trait à la météorologie. Enfin, je déroule les étapes de la méthode.

Requête de l'utilisateur Dans notre exemple, l'utilisateur est notre chercheur de Lyon qui souhaite obtenir des données de rayonnement UVA entre le 1er janvier 2008 et le 10 janvier 2008 à Paris, dont les coordonnées géographiques sont $(48, 2)$.

Pour utiliser notre méthode, l'utilisateur doit traduire sa requête dans un langage compréhensible par le prototype. Dans notre cas l'utilisateur est un utilisateur direct et avancé de la méthode de composition, dans le cas général l'utilisateur n'aura pas connaissance du fonctionnement de la méthode de composition. Son rôle sera de remplir un formulaire, la requête pouvant être pré-établie par un professionnel du domaine connaissant la méthode et les requêtes types de ces utilisateurs. Il commence par définir les types des données qu'il va fournir ainsi que les types des données qu'il souhaite obtenir par :

```
Requete : (geopoint, period) -> irradiance
```

où :

- `geopoint` est le type représentant les coordonnées géographiques ;
- `period` est le type représentant l'intervalle de temps désiré par l'utilisateur ;
- `irradiance` est le type des données que souhaite obtenir l'utilisateur.

L'utilisateur doit ensuite définir la composition qu'il souhaite de façon plus précise afin de permettre à la méthode d'éliminer les compositions invalides.

Considérons que l'utilisateur souhaite réaliser la composition suivante :

```
prog
    func getIrradUVA exec
        tmp = { final : "irradUVA" } ;
        tmp.date = date;
        tmp.geopoint = geopoint;
        tmp.unit = unit
    return tmp
    func ServiceA exec
        result = [ ];
```

```
        foreach i in range(period.begin, period.end) do
            tmp = { sequence : [ ] };
            tmp.date = i;
            tmp.geopoint = geopoint;
            tmp.unit = "JM2";
            result.sequence = append(result.sequence, getIrradUVA (tmp))
        done
    return result
    none
exec
    request = { latitude : 48, longitude : 2 };
    request.period = { begin : 20080101, end : 20080110 };
    result = ServiceA(request);
return result
```

Dans cette composition, l'utilisateur déclare un service Web : getIrradUVA. L'identifiant clef *final* contenu dans la partie résultat indique que ce service Web est un service Web de base. L'utilisateur déclare aussi un service Web normal nommé ServiceA. A cette étape, il n'est pas nécessaire que ServiceA soit effectivement un service Web existant.

Base de données des services Web connus Pour cet exemple, le prototype connaît les N services Web suivants :

- ServiceB qui fournit des données de rayonnement global total à partir d'un lieu donné ;
- ServiceA qui fournit le rayonnement dans les UVA pour un lieu particulier et une période donnée ;
  Les deux services Web précédents fournissent les données en Jm^{-2} ;
- on dispose ensuite d'un service Web, Service C, qui permet, à partir du rayonnement global total, de fournir des rayonnements UVA, obtenus par un calcul approprié ;

De la même façon que l'utilisateur définit sa requête, les services Web décrivent les types de leurs entrées et de leurs sorties.

La description des types d'entrée et de sortie des services Web, donnée ci-dessous, s'écrit de la façon suivante :

$ServiceX : (I_1, I_2) -> O$

Et se lit : le $ServiceX$ a pour entrée des données de type I_1 et I_2, et a pour sortie une donnée de type O.

Voici la description des services Web :

```
Service A : (geopoint,period) -> (irradiance)
Service B : (geopoint,period) -> (irradiance)
Service C : (irradiance) -> (irradiance)
```

Comme nous pouvons le remarquer, cette description ne permet pas de différencier tous les services Web entre eux. En effet, si on se limite à l'analyse de cette description, les services Web A et B font la même chose. Ceci illustre la raison pour laquelle certaines méthodes ne discernent pas certains services Web entre eux et que la description comportementale est importante.

Pour décrire le comportement des services Web, il faut commencer par décrire les fonctions de référence. Ces fonctions permettent de fournir les données de base qui seront manipulées par les services Web. Dans notre exemple, nous avons deux types de base dépendants, qui sont irradGlobalTotal et irradUVA.

Ces types de base sont décrits dans le langage de la façon suivante :

```
func getIrradGlobalTotal exec
    tmp = { final : "irradGlobalTotal" } ;
    tmp.date = date;
    tmp.geopoint = geopoint;
    tmp.unit = unit;
return tmp

func getIrradUVA exec
    tmp = { final : "irradUVA" } ;
    tmp.date = date;
    tmp.geopoint = geopoint;
    tmp.unit = unit;
return tmp
```

Ces types d'irradiance dépendent uniquement d'une date, d'une localisation et de leurs unités.

La différenciation entre les deux types est faite par la présence du mot clef *final* dans le résultat, le premier service de base fournit le mot clef $final$ associé à "$irradGlobalTotal$", alors que le second associe "$irradUVA$" à l'identifiant clef final. Ceci permettra de les différencier lors de l'exécution théorique.

Comme il est impossible de déterminer les relations entre ces différents types par cette unique description, il est nécessaire de définir le comportement des services Web. Ce comportement va déterminer comment les services Web produisent et utilisent les données des types de base.

La définition des services Web présentés précédemment est la suivante :

```
func ServiceA exec
    result = { sequence : [ ] };
    foreach i in range(period.begin, period.end) do
        tmp = { } ;
        tmp.date = i;
        tmp.geopoint = geopoint;
        tmp.units = "JM2";
        result.sequence = append(result.sequence, getIrradUVA (tmp))
    done
return result
```

On remarque que le service Web ServiceA commence par initialiser le résultat, appelé *result*, comme un dictionnaire comportant une liste vide, appelée *sequence*. Ensuite il parcourt l'intervalle temporel entre *period.begin* et *period.end*. À chaque

date i, il appelle le service Web de base getIrradUVA au geopoint donné et à la date considérée. De plus, il définit l'unité de chaque valeur comme "$JM2$". Son résultat est construit par la concaténation des résultats renvoyés par le service Web.

```
func ServiceB exec
    result = { sequence : [ ] };
    foreach i in range(period.begin, period.end) do
        tmp = { };
        tmp.date = i;
        tmp.geopoint = geopoint;
        tmp.unit = "JM2";
        result.sequence = append(result.sequence, getIrradGlobalTotal (tmp))
    done
return result
```

On remarque que la description de ce ServiceB est identique à celle de ServiceA sauf que ServiceB fait appel à getIrradGlobalTotal à la place de getIrradUVA ; ceci signifie que ServiceB fournit une série temporelle d'irradiance globale totale.

Enfin ServiceC parcourt la liste *sequence* qui lui est fournie en entrée de manière à y repérer par un test les valeurs de l'irradiance globale totale. ServiceC change le type de chaque valeur finale trouvée en $irradUVA$, ce qui peut être interprété comme une conversion d'une irradiance globale totale en une irradiance dans les UVA. On remarquera également que le service Web élimine toutes les autres valeurs possibles et renvoie une liste ne comportant que des valeurs d'irradiance dans les UVA.

```
func ServiceC exec
    result = { sequence : [ ] };
    foreach i in sequence do
        if i.final == "irradGlobalTotal" then
            i.final = "irradUVA";
            result.sequence = append(result.sequence, i)
        else
            skip
        end
return result
```

Application et déroulement de la méthode proposée La première étape consiste à trouver les chemins dans le graphe formé avec la représentation conventionnelle. Pour notre exemple, on obtient les chemins suivants :

ServiceA ;
ServiceB ;
ServiceA , ServiceC ;
ServiceB , ServiceC ;

Une fois tous les chemins possibles sans boucle définis, il faut traduire ces chemins dans le langage décrivant le comportement des services Web. Par exemple, le chemin Service A, Service C sera traduit par :

```
prog
    ... déclaration des services Web ...
exec
    input0 = { geopoint : { latitude : 48, longitude : 2 } };
    input0.period = { begin : 20080101, end : 20080110 };
    input1 = ServiceA (input0);
    input2 = ServiceC (input1)
return input2
```

Chacun de ces programmes est exécuté pour obtenir l'environnement final qui sera comparé à l'environnement souhaité. La règle appliquée est *Si la valeur de retour est incluse dans la valeur de retour souhaitée par l'utilisateur, alors cette composition est valide* ; cette règle est réalisée par la comparaison des environnements. L'étape finale consiste à exécuter la composition choisie. Voici le résultat obtenu pour l'exécution de la requête de l'utilisateur :

```
{ sequence :
        { final : "irradUVA", date : 20080101,
        geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
        { final : "irradUVA", date : 20080102,
        geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
        { final : "irradUVA", date : 20080103,
        geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
        ...
        { final : "irradUVA", date : 20080110,
        geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
}
```

Puis voici le résultat de l'appel à ServiceB :

```
{ sequence :
        { final : "irradGlobalTotal", date : 20080101,
        geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
        { final : "irradGlobalTotal", date : 20080102,
        geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
        { final : "irradGlobalTotal", date : 20080103,
        geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
        ...
        { final : "irradGlobalTotal", date : 20080110,
        geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
}
```

On constate que, dans cette composition, le résultat n'est pas conforme à la requête ; cette séquence sera donc éliminée. Les compositions invalides sont éliminées et les compositions restantes sont triées par ordre de qualité globale. Il ne reste plus que les compositions suivantes dans l'ordre de qualité :

ServiceA ;
ServiceB , ServiceC ;

Supposons que le résultat obtenu par la première composition soit :

{ sequence :
{ value : 400, date : 20080101,
geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
{ value : 410, date : 20080102,
geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
{ value : 405, date : 20080103,
geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
}

On n'obtient que les trois premières valeurs souhaitées : les trois premiers jours de janvier 2008. La méthode ne validera donc pas ce résultat et essayera la séquence suivante :

{ sequence :
{ value : 405, date : 20080102,
geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
{ value : 408, date : 20080103,
geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
...
{ value : 500, date : 20080110,
geopoint : { latitude : 48, longitude : 2 }, unit : "JM2" }
}

Le résultat est plus complet mais il manque la première valeur. Le prototype fera donc l'union de ces résultats de manière à fournir un résultat complet à l'utilisateur.

5.8 Cas particulier où la comparaison des comportements échoue

Dans certains cas, la simulation de la composition peut échouer là où la composition réelle pourrait fonctionner. Par exemple, lorsque l'opération **foreach** s'applique à l'appel d'un service Web de base MonServiceDeBase :

```
foreach i { MonServiceDeBase (i) }
{
```

```
  ...
}
```

La raison pour laquelle la méthode de composition proposée ne peut pas évaluer cette composition est évidente ; étant donné qu'elle ne connaît pas la liste de retour possible de MonServiceDeBase, elle ne peut pas l'évaluer et parcourir cette liste. Le même constat peut être fait avec les opérateurs **if ... then ... else**, lorsque la condition du if dépend du résultat d'une fonction indéfinie.

Une façon de réaliser la vérification consisterait à évaluer le service Web permettant d'obtenir la valeur de cette information, ainsi la formule pourrait être évaluée et comparée. L'avantage de cette méthode est qu'elle accroît les chances de trouver une composition valide. En contre-partie, les performances en sont diminuées car des requêtes plus nombreuses seront effectuées sur les services Web sans garantie que ces requêtes soient utiles.

5.9 Analyse de la méthode proposée

La méthode réalise en partie les formes de compositions énoncées dans le chapitre 3. Par exemple, elle réalise pleinement la séquence alors qu'elle ne réalise que partiellement l'union et le choix.

Le succès de cette méthode dépend principalement de la capacité d'abstraction de l'expert qui devra décrire son service Web. Plus il sera précis et exact, plus le service Web sera utile pour la composition.

Cette méthode n'est naturellement pas complète simplement parce que des méthodes complètes n'existent pas encore pour les cas généraux. En effet, comme on l'a vu, le nombre des compositions possibles et valides peut parfois être infini et il est impossible d'exprimer une infinité de solutions et de les exploiter. Si l'on prend la caractéristique d'Internet, personne n'est prêt à attendre très longtemps une réponse lors de l'interrogation d'un service Web.

Relation de cette méthode avec les descriptions de services Web existantes ou à venir. Cette méthode peut être reliée à la description OWL-S utilisant les données du *ServiceProfile*. Les pré- et post- conditions peuvent être utilisées à la première étape de la méthode de composition alors que les *ProcessModel* peuvent être utilisés à la seconde étape. Le langage peut être mis en correspondance avec le langage OWL grâce au tableau suivant :

- Sequence : séquence du langage
- Split : peut être traduit par des séquences
- Split+join : peut être traduit par des séquences
- Any-order : peut être traduit par des séquences
- Choice : n'est pas inclus directement dans le langage
- If-Then-Else : **if ... then ... else ... end** du langage
- Iterate : forme de boucle **foreach ... in ... do ... done**
- Repeat-While et Repeat-Until : ne peuvent pas être réalisés car récursif avec possibilité de non-terminaison.

La traduction n'est pas directe car le langage que je propose est plus contraignant. Néanmoins, la méthode proposée n'en est pas moins valide et montre une nouvelle façon d'utiliser la description de type OWL-S. Une différence importante est celle

liée aux fonctions de base, notion qui n'est pas utilisée par OWL-S. Ces fonctions de base peuvent simplement être vues comme des constructeurs d'objets et donc des objets typés. L'avantage de ces fonctions est qu'elles sont plus générales que les types traditionnels et permettent de définir des fonctions de base dont le but n'est pas de créer directement de nouvelles données, par exemple une fonction calculant un modulo. Il est possible ensuite d'introduire de nouvelles règles permettant d'étendre l'équivalence. Par exemple, définir une fonction de base add, définissant le mot clef $final$ à "*add*", avec la propriété suivante : add(a,b) est équivalent à add(b,a)

Précision de la description basée sur les types des entrées/sorties Il serait possible d'augmenter la précision de la description des types des entrées et des sorties en y incluant des types plus précis comme irradUVA, irradUVB et irradGlobalTotal, mais il apparaît alors des difficultés pour décrire les services Web de conversion générique d'unité par exemple ; ou bien, on élimine simplement la possibilité d'utiliser ces services, car ils ne seraient plus compatibles avec les autres services Web fournissant des données. Une solution plus élaborée consiste à définir des relations plus complexes entre les différents types de données manipulées. Il existe pour le Web trois groupes de relations standardisées par le W3C qui sont plus ou moins précis : OWL Lite, OWL DL et OWL Full dans l'ordre croissant de complexité. OWL Lite permet de classer des éléments avec des contraintes simples, il est limité pour permettre une utilisation facile. OWL DL, quant à lui, offre l'expressivité maximum tout en garantissant la calculabilité et la décidabilité. Et enfin, OWL Full offre la plus grande expressivité au détriment de la calculabilité et de la décidabilité [73].

Ces langages, dans certaines limites, permettent d'établir des relations entre les types des entrées et des sorties des services Web, plus expressives que celles utilisées actuellement dans cette méthode. En effet, dans cette méthode, les relations entre les types sont de un à un ; par exemple "irradUVA" n'est compatible qu'avec d'autres "irradUVA". Cependant, il est possible d'intégrer une comparaison de type qui intégrerait des notions de sous-ensemble, comme des héritages qui hiérarchisent les types entre eux. Pour l'intégrer, cette méthode devrait disposer d'une fonction lui permettant de déterminer si deux types sont compatibles et qui servirait à comparer les données de l'identifiant clef $final$.

Un langage Mon idée initiale était d'utiliser le langage d'une façon différente de celle proposée dans ce document. Elle consistait à définir un langage permettant d'exprimer des classes d'équivalence entre différents schémas de codes. Il existe deux types d'équivalence de schémas : le premier est celui interne au langage de programmation, ce sont des propriétés du langage ; le deuxième est la déclaration explicite d'équivalence, par exemple, telle fonction de base A est équivalente à telle autre fonction B. Puis cette méthode aurait utilisé les propriétés d'équivalence du langage pour réécrire les compositions en d'autres, ces nouvelles réécritures ayant la caractéristique de n'utiliser que des fonctions qu'on sait pouvoir réaliser avec un ou plusieurs services Web.

J'ai rapidement abandonné cette piste car elle était vouée à l'échec, étant donné le caractère non déterministe de la réécriture. Pour illustrer ce problème, supposons qu'il soit possible de réécrire la phrase AB en ACB et que l'on puisse réécrire CB en BC. La phrase AB peut alors être réécrite d'une infinité de façons différentes, par exemple ACBBB, et il est impossible de savoir quelles sont les formes de composition atteignables et quelle réécriture est plus intéressante à utiliser en premier. Pour

résoudre cette difficulté, j'ai ensuite étudié la possibilité de faire intervenir de l'apprentissage, issu de l'intelligence artificielle, en liant des situations types aux réécritures qui semblent les plus utiles. J'ai choisi de ne pas poursuivre dans cette voie, car une solution moins risquée selon moi, et plus raisonnable est apparue. Il serait probablement intéressant d'étudier cette voie plus en avant, afin de voir si une heuristique convergente peut être établie dans le domaine de la météorologie.

La méthode hybride est intéressante pour plusieurs raisons. D'abord, elle utilise des descriptions proches de OWL-S qui semble émerger comme le prochain standard de description des services Web. Elle combine les descriptions des entrées et des sorties avec une description comportementale, elle tire donc partie des avantages des méthodes de composition basées sur la planification et des avantages des méthodes utilisant la preuve de programme. Ainsi, la méthode se termine toujours. De plus, si la méthode fournit une bonne réponse, alors cette réponse est probablement raisonnable. Néanmoins, cette méthode est également limitée car elle ne peut proposer, explorer toutes les solutions. En comparaison avec d'autres méthodes, elle est proche de la solution proposée par Medjahed [66] tout en l'améliorant par des approches comme celles utilisées par Rao [80] ou Wanermman [104] basées sur la preuve de programme. L'un des intérêts majeurs de cette méthode est qu'elle permet d'évaluer virtuellement une composition. Ceci permet de simuler certaines unions possibles entre les compositions, ou d'évaluer la qualité de chacune des données indépendamment de l'exécution du service Web. Enfin, la description comportementale permet aussi d'évaluer si le service Web fournit les données décrites ou non, en particulier les séries temporelles qui sont complètement décrites.

Limite de l'implémentation Les services Web utilisés ne suivent pas le protocole standard habituel qui est SOAP. La principale raison est d'ordre pratique : en effet, il existe en Python une librairie implémentant XML/RPC qui présente l'avantage d'être simple à mettre en oeuvre. De plus, les données transmises sont des dictionnaires du langage Python, structurés à la manière des objets SOAP. Le dictionnaire de base représente un message SOAP comportant les paramètres d'appel ou de retour du service Web. Chaque paramètre est structuré à l'aide de dictionnaires, de listes et de types de base. Bien que cette définition soit différente, l'approche est la même qu'avec les services Web standards : chaque service Web est défini à l'aide d'une signature typée. La réalisation de services Web utilisant des protocoles habituels tels que SOAP n'est pas simple mais ne présente pas de difficulté scientifique.

Chapitre 6

Évaluation des méthodes de composition automatique et adaptative

Après la présentation dans le chapitre précédent des différentes méthodes de composition que j'ai mises en oeuvre, ce chapitre spécifie, implémente et analyse un outil d'évaluation des méthodes de composition vis-à-vis des améliorations attendues en météorologie. Notre but est de proposer une méthode qui s'applique à n'importe quelle méthode de composition adaptative et automatique.

Analyser et comparer ces différentes méthodes de composition revient à analyser et comparer la pertinence des méthodes utilisées pour réaliser la fonction de composition Fc décrite dans le chapitre 2.

Une évaluation exhaustive ou théorique de cette fonction ne serait possible qu'en présence de tous les éléments théoriques, comme, par exemple une formalisation mathématique de la méthode employée pour réaliser la composition. Or, dans le cas général, nous ne disposons pas de ces éléments. J'ai donc choisi de fonder ma méthode d'évaluation sur l'analyse d'un ensemble de cas représentatifs que j'appelle scénarios et avec lesquels je vais évaluer le comportement des différentes méthodes.

Les scénarios sont construits de manière à représenter le plus fidèlement possible les besoins exprimés en météorologie. Chaque scénario réalisé doit permettre de discuter des capacités de la méthode testée. L'outil d'évaluation va donc établir, pour chaque méthode testée, une fiche qui synthétise les différents comportements constatés.

Dans la suite de ce chapitre, je commence par définir les formes des compositions types représentatives des compositions que les météorologues souhaitent réaliser et dont une notion principale est la notion de qualité, et je caractérise leur environnement d'exécution.

Ensuite, je définis les scénarios et la méthode d'évaluation de ces scénarios de manière à déterminer les comportements des scénarios.

Puis je décris l'implémentation que j'ai mise en œuvre de cet outil d'évaluation. J'utilise ensuite cet outil pour évaluer et comparer les prototypes réalisés, ainsi que la nouvelle méthode de composition que je propose dans les chapitres 4 et 5. Je présente les résultats des tests et je termine par une analyse de l'outil lui-même.

6.1 Analyse des besoins pour l'évaluation

Dans cette section, j'analyse les différents aspects permettant d'établir une méthode d'évaluation des méthodes de composition. Une première partie établit les formes de composition candidates à être testées. Puis nous étudions la capacité des méthodes à prendre en compte la qualité des services Web. Enfin, nous analysons l'environnement des compositions, c'est-à-dire les services Web qui devront être simulés lors des tests.

6.1.1 Les formes de compositions à évaluer

Évaluer une méthode de composition, revient à vérifier si cette méthode est capable de combiner les services Web de façon pertinente dans une situation donnée. La pertinence d'une composition est établie par la distance entre la réponse obtenue et la réponse attendue : plus la réponse obtenue sera proche de la réponse attendue, plus la composition sera pertinente.

Plusieurs formes de composition me semblent représentatives des compositions que souhaitent réaliser les météorologues. Un premier ensemble de formes de composition de base a été défini dans le chapitre 3. Je rappelle ces formes ci-dessous, spécifiées dans le langage définis dans le chapitre 3, par l'équation référencée par (6.x) :

- *appel d'un service Web unique "S_a"* (6.1) ;
- *séquence de deux services Web, "S_b puis S_c"* (6.2) ;
- *choix exclusif entre deux services Web, "S_d ou S_e"* (6.3) ;
- *union des résultats de deux services Web, "S_f et S_g"* (6.4).

Se limiter à l'évaluation de ces formes de base n'est pas suffisant puisqu'elles ne permettent pas d'évaluer la capacité des méthodes de composition à combiner ces formes de composition entre elles.

Pour cette raison, je propose un nouvel ensemble de formes de compositions qui sont des combinaisons des formes de base précédentes. Il sera ainsi plus difficile, pour une méthode, de *passer* les tests de façon opportuniste et l'évaluation gagnera en précision. Je donne ci-dessous les formes qui m'ont semblé simples et pertinentes, spécifiées dans le langage définis dans le chapitre 3 par l'équation référencée par (6.x) :

- *choix entre un service et une séquence de services, "(S_h et (S_i puis S_j))"* (6.5) ;
- *choix entre deux séquences de deux services Web, "(S_k puis S_l) et (S_m puis S_n)"* (6.6) ;
- *union de l'appel d'un service Web et d'une séquence de deux services Web, "(S_o et (S_p puis S_q))"* (6.7) ;
- *union des résultats de deux séquences de deux services Web, "S_r puis S_s et S_t puis S_u"* (6.8) ;
- *séquence entre l'union de deux services Web "(S_v et S_w) puis appel de S_x"* (6.9).

Comme on le constate, ces formes combinent le choix et la séquence ou l'union et la séquence. J'ai choisi de ne pas évaluer des combinaisons de choix et d'union car, comme nous le verrons par la suite, ces deux formes de composition sont très proches.

La dernière forme 6.9 est particulière car elle vise à tester la capacité d'utiliser deux services Web pour en appeler un troisième. Ceci exprime, par exemple, le fait que les services Web S_v et S_w ne fournissent pas des données de même nature (par exemple, l'un d'eux fournit une série temporelle du trouble de Linke, et l'autre, une série temporelle de rayonnement solaire hors atmosphère).

$$S_a \tag{6.1}$$
$$S_c \circ S_b \tag{6.2}$$
$$S_d \oplus S_e \tag{6.3}$$
$$S_f \uplus S_g \tag{6.4}$$
$$S_h \oplus (S_j \circ S_i) \tag{6.5}$$
$$(S_l \circ S_k) \oplus (S_n \circ S_m) \tag{6.6}$$
$$S_o \uplus (S_q \circ S_p) \tag{6.7}$$
$$(S_s \circ S_r) \uplus (S_u \circ S_t) \tag{6.8}$$
$$S_x \circ (S_v \uplus S_w) \tag{6.9}$$

Notons que ces formes seront utilisées plusieurs fois pour définir les différents scénarios.

6.1.2 Évaluation de la qualité

Nous avons vu par ailleurs, qu'en météorologie, il est possible d'estimer une incertitude sur les données et donc de prendre en compte cette incertitude dans le choix de la composition. Cette prise en compte de la qualité lors de la composition est spécifique aux services Web en météorologie. Elle peut plus généralement se retrouver dans les géosciences. Elle influe sur le comportement souhaité de la méthode de composition. Normalement, la qualité devrait être prise en compte après la disponibilité. Par conséquence, lorsque plusieurs compositions sont possibles et que chacune d'elles fonctionne, le choix de la composition doit porter sur une estimation de la qualité des données. Pour cette raison, je fournis un ensemble de scénarios dont l'objectif est de vérifier que les méthodes de composition prennent en compte cette qualité.

Étant donné que la qualité ne fait pas partie de la description standard des services Web (cf. [20, 110]), les méthodes pourront la définir elles-mêmes. De plus, cet outil ne se base pas sur les qualités absolues des données, mais sur une qualité décrite. En d'autres termes, la véracité des descriptions n'est pas remise en question, contrairement à certaines parties des descriptions portant sur le fonctionnement des services Web.

6.1.3 Les environnements des compositions

Une fois définies les capacités des formes de composition que je souhaite évaluer, il faut caractériser les environnements auxquels sont soumis les méthodes de composition, de manière à pouvoir les simuler.

Le premier élément dont dépendent les compositions de services Web sont les services Web eux mêmes. Je les analyse en premier. Internet constitue le deuxième élément à prendre en compte, je définis donc ensuite la notion de contexte Internet en différenciant contexte Internet connu et contexte Internet réel.

6.1.3.1 L'état d'un service Web

La description habituelle des services Web est incomplète [66] ou abstraite, ce qui laisse une certaine latitude dans l'interprétation de cette description. Par exemple,

lorsqu'un service affirme qu'il fournit une série temporelle à partir de coordonnées géographiques, il ne donne pas nécessairement d'indication sur la zone géographique et temporelle qu'il couvre ; de même, il est possible de prétendre qu'un service Web fonctionne alors qu'il est tombé en panne. Cet écart entre description et réalité permet de distinguer trois types de services Web. Les services Web peuvent donc être dits :

- **parfaits** : les services web parfaits disposent d'agents qui fournissent toujours la donnée souhaitée correspondant aux entrées fournies. Par exemple, un service Web fournissant une conversion d'unité est le plus souvent parfait, il renvoie toujours la valeur convertie ; ces services Web fournissent les données décrites ;
- **normaux** : sont des services Web dont les agents ont un comportement *aléatoire* du point de vue de l'utilisateur, car ils fournissent des données parfois incomplètes. Par exemple, un service Web qui affirme fournir des séries temporelles, peut fournir des séries incomplètes. Cette incomplétude n'est pas reportée dans la description du service Web ;
- **en panne** : sont des services Web dont les agents ne répondent pas aux requêtes, ou sont inexistants.

J'ai choisi de distinguer les services Web parfaits pour disposer d'un outil d'évaluation plus complet et plus fin. En faisant cette distinction, l'outil pourra évaluer le comportement de la méthode de composition dans une situation favorable avec des services Web toujours disponibles et parfaitement décrits, et dans une situation plus réaliste dans laquelle les services Web seront normaux. Cette distinction est également utile, compte-tenu des méthodes habituellement utilisées pour le développement de logiciel. Le développement s'effectue souvent de manière à réaliser une tâche dans une situation favorable, puis, dans un second temps, à prendre en compte les cas défavorables ou peu représentés.

Par ailleurs, je distingue les services Web en panne, pour des raisons de granularité ; de cette façon l'outil pourra évaluer la possibilité de remplacer un service Web par un autre indépendamment de la possibilité de faire l'union de données en provenance de plusieurs services Web.

6.1.3.2 Le contexte Internet

Le contexte Internet se rapporte à l'état des réseaux, des services Web et des autres ressources d'Internet comme les sites Web ou les serveurs ftp. Ce contexte est dynamique et très évolutif : des services Web peuvent y être ajoutés, d'autres supprimés, le réseau peut être encombré, ou partiellement détruit. Je définis donc le contexte Internet par l'union des états des ressources Internet à un moment donné ; dans notre cas, les ressources qui nous intéressent sont les services Web. Comme il est difficile de connaître l'état d'Internet en temps réel, et à plus forte raison de toutes les ressources Internet, la définition du contexte Internet se limite généralement à quelques ressources. Le fait que les contextes ne concernent qu'un nombre limité de ressources, implique que la comparaison de deux contextes n'est pertinente que si ces deux contextes se rapportent aux mêmes ressources. Il sera nécessaire pour les scénarios de définir les contextes Internet utilisés, afin de déterminer les résultats attendus, ces résultats dépendant directement des ressources Internet disponibles.

6.2 La méthode d'évaluation

6.2.1 Les différents éléments des scénarios

L'outil d'analyse de la pertinence des méthodes de composition définit pour chaque scénario, les éléments suivants :

1. la forme de composition du scénario, chaque scénario étant issu d'une des formes de composition exposées dans la section 6.1.1 ;
2. le contexte Internet connu par la méthode de composition, qui est composé des descriptions des services Web connus ;
3. le contexte Internet réel, dans lequel la méthode va être exécutée, qui est composé des services Web réellement disponibles et peut donc exhiber des services Web en panne ;
4. la requête, fournissant les données pour réaliser la composition, qui définit les données fournies et les données souhaitées par l'utilisateur ;
5. le résultat attendu, compte-tenu du contexte Internet réel, qui est défini par chaque scénario. La définition de ce résultat permettra d'évaluer la pertinence de chacune des méthodes testées vis-à-vis des améliorations attendues.

Notons que dans la définition des scénarios, les deux contextes connus et réels d'Internet sont définis par les mêmes ressources de manière à pouvoir les comparer. Notons aussi que les services Web non cités par ces contextes sont considérés comme inexistants dans le contexte considéré, à la manière des mondes clos de la planification.

6.2.2 Étapes de définition des scénarios

La définition de chaque scénario se déroule en trois étapes :

1. une définition générale, fixant la forme de composition testée, le contexte Internet connu et le contexte Internet réel ;
2. la définition de l'ensemble des services Web entrant dans la réalisation de ce scénario ;
3. la définition de la requête de l'utilisateur et le résultat attendu, compte-tenu du contexte Internet réel.

Ces trois étapes sont nécessaires et s'expliquent par le fait que la requête de l'utilisateur et le résultat en découlant, dépendent des services Web qui réaliseront les scénarios. De plus, pour pouvoir définir les services Web nécessaires, il faut connaître les formes de scénarios testés ainsi que les contextes Internet, comme nous le verrons par la suite.

6.2.3 Décomposition des scénarios

Chaque scénario se décompose en deux parties : la composition que la méthode doit réaliser et la liste de services Web connus par la méthode. Notons que, par défaut, la méthode connaît tous les services Web de la composition à réaliser, ces services Web pouvant être parfaits, en panne ou normaux.

6.2.4 Définition des scénarios

Comme il a été indiqué dans la section précédente, je ne définis ici que les formes de composition et les contextes Internet de chaque scénario. Les scénarios sont représentés à l'aide du langage formalisant les compositions et définis dans le chapitre 3 auquel j'ajoute les deux modificateurs d'état des services Web $\top$ et $\bot$. Ces modificateurs indiquent l'état réel du service Web lors de l'exécution de la requête :

- $\top S$ signifie que l'opération Sdu service Web est parfaite ;
- $\bot S$ signifie que l'opération S de service Web est en panne ;
- S l'absence de modificateur signifie que l'opération S est normale.

Par exemple, le scénario suivant

$$(S_s \circ \top S_r) \uplus (S_u \circ \top S_t)$$

indique que :

- S_r, S_s, S_t et S_u sont des opérations de service Web connues de la méthode de composition ;
- les opérations S_s et S_u seront normales au moment de la requête ;
- S_r et S_t seront des opérations parfaites ;
- il est possible d'enchaîner les opérations S_r puis S_s et les opérations S_t puis S_u ;
- il est possible de faire l'union des deux séquences précédentes.

Je présente dans le tableau 6.1 ci-dessous, les différents scénarios de référence à évaluer.

Nom du scénario	Scénario
sc_appel_A	$\top S_a$
sc_appel_B	S_a
sc_appel_C	$\bot S_a$
sc_seq_A	$\top S_c \circ \top S_b$
sc_seq_B	$S_c \circ S_b$
sc_seq_C	$S_c \circ \bot S_b$
sc_choix_A	$\top S_d \oplus \top S_e$
sc_choix_B	$S_d \oplus S_e$
sc_choix_C	$S_d \oplus \bot S_e$
sc_union_A	$\top S_f \uplus \top S_g$
sc_union_B	$S_f \uplus S_g$
sc_union_C	$S_f \uplus \bot S_g$
sc_choix1_A	$\top S_h \oplus (\top S_j \circ \top S_i)$
sc_choix1_B	$S_h \oplus (S_j \circ S_i)$
sc_choix1_C	$S_h \oplus (\bot S_j \circ S_i)$
sc_choix2_A	$(\top S_l \circ \top S_k) \oplus (\top S_n \circ \top S_m)$
sc_choix2_B	$(S_l \circ S_k) \oplus (S_n \circ S_m)$
sc_choix2_C	$(S_l \circ S_k) \oplus (\bot S_n \circ S_m)$
sc_union1_A	$\top S_o \uplus (\top S_q \circ \top S_p)$
sc_union1_B	$S_o \uplus (S_q \circ S_p)$
sc_union1_C	$S_o \uplus (\bot S_q \circ S_p)$
sc_union2_A	$(\top S_s \circ \top S_r) \uplus (\top S_u \circ \top S_t)$
sc_union2_B	$(S_s \circ S_r) \uplus (S_u \circ S_t)$
sc_union2_C	$(S_s \circ S_r) \uplus (\bot S_u \circ S_t)$
sc_seq1_A	$\top S_x \circ (\top S_v \uplus \top S_w)$
sc_seq1_B	$S_x \circ (S_v \uplus S_w)$
sc_seq1_C	$S_x \circ (S_v \uplus \bot S_w)$

TAB. 6.1 – Tableau de référence des différents scénarios à évaluer

6.2.5 Les procédures de test des scénarios

Pour chacune des méthodes de composition du chapitre 5, je n'exécute qu'une seule fois les tests car les méthodes testées sont déterministes.

Notons que si nous utilisions des méthodes basées sur l'apprentissage et l'intelligence artificielle, méthodes qui, étant capables d'apprendre, pourraient changer leurs résultats au fil du temps, il serait alors utile d'exécuter plusieurs fois les tests des méthodes à cause de l'indéterminisme engendré par l'apprentissage.

L'évaluation des méthodes de composition analyse leurs fonctionnalités et leur qualité, je présente ces deux aspects dans les sections suivantes.

6.2.5.1 Exécution des scénarios

L'exécution d'un scénario consiste à ;

- démarrer les services Web qui sont utilisés par le scénario de manière à les rendre disponibles. Les autres demeurent arrêtés ;
- lancer l'exécution de la méthode de composition qu'on souhaite tester avec la requête et les données utilisateur ;
- récupérer le résultat de l'exécution dans le but de le comparer avec le résultat idéal attendu.

6.2.5.2 Evaluation des fonctionnalités

Les fonctionnalités testées sont décrites dans la section 6.1.1, et consistent, par exemple, à évaluer la capacité de choisir entre deux compositions, ou encore de faire l'union de deux compositions, c'est-à-dire, les améliorations souhaitées par les météorologues ne concernant pas la qualité.

L'évaluation des fonctionnalités consiste à établir un tableau qui synthétise les résultats obtenus après les exécutions de la méthode de composition à évaluer sur les différents scénarios. Ce tableau est composé de cinq colonnes. La première indique le nom du scénario, les quatre suivantes sont dans l'ordre :

- la première colonne rapporte si le résultat de l'exécution du scénario est complet ;
- la deuxième colonne rapporte si le résultat de l'exécution du scénario est partiel (il manque au moins une donnée par rapport au résultat complet) ;
- la troisième colonne indique si le résultat de l'exécution du scénario est vide (aucune donnée n'a été obtenue) ;
- la quatrième colonne indique si le résultat du scénario est faux (ce résultat comporte une ou plusieurs données qui n'existent pas dans le résultat complet comme par exemple des données en degré Celsius à la place de données en degré Farenheit).

où le résultat complet d'un scénario est un résultat théorique spécifié à la main ; c'est par définition le résultat qu'aurait obtenu une méthode idéale avec tous les services Web du scénario parfaits à partir duquel on va en déterminer les résultats incomplets ou faux.

Le tableau 6.2 montre un exemple de l'exécution de tous les scénarios par une méthode donnée. Ce tableau est également le tableau, qu'aurait obtenu la méthode idéale (sans bien sûr que les services Web soient tous parfaits). Il servira de référence pour analyser les tableaux obtenus par les méthodes réelles de composition. En comparant ce tableau (6.2) avec le tableau obtenu par une méthode quelconque, il est possible de connaître les fonctionnalités que cette dernière réalise. Par exemple, si

les lignes `seq_A`, `seq_B`, `seq_C` sont identiques au tableau de référence 6.2 pour une méthode donnée, on en conclut que cette méthode réalise les compositions comportant des séquences services Web. Noter le fait que, dans certains scénarios, le résultat souhaité est l'absence de résultat. C'est le cas, par exemple, pour le scénario `appel_C` où le service Web est indisponible. En effet, il n'est pas raisonnable qu'une méthode fournisse des résultats alors qu'aucun service Web n'est disponible.

	1	**2**	**3**	**4**
sc_appel_A	1	0	0	0
sc_appel_B	0	1	0	0
sc_appel_C	0	0	1	0
sc_seq_A	1	0	0	0
sc_seq_B	0	1	0	0
sc_seq_C	0	0	1	0
sc_choix_A	1	0	0	0
sc_choix_B	0	1	0	0
sc_choix_C	0	1	0	0
sc_union_A	1	0	0	0
sc_union_B	1	0	0	0
sc_union_C	0	1	0	0

	1	**2**	**3**	**4**
sc_choix1_A	1	0	0	0
sc_choix1_B	0	1	0	0
sc_choix1_C	0	1	0	0
sc_choix2_A	1	0	0	0
sc_choix2_B	0	1	0	0
sc_choix2_C	0	1	0	0
sc_union1_A	1	0	0	0
sc_union1_B	1	0	0	0
sc_union1_C	1	0	0	0
sc_union2_A	1	0	0	0
sc_union2_B	1	0	0	0
sc_union2_C	0	1	0	0
sc_seq1_A	1	0	0	0
sc_seq1_B	0	1	0	0
sc_seq1_C	0	0	1	0

TAB. 6.2 – Tableau de référence pour un test unique réalisé avec une méthode idéale. La colonne 1 indique si le test a réussi ; la colonne 2 rapporte si le test est partiellement réussi (il manque au moins une donnée) ; la colonne 3 indique si le test a échoué (aucune donnée n'a été fournie) ; la colonne 4 indique si le résultat est faux (résultat comportant des données fausses).

Pour réaliser une analyse plus fine des méthodes de composition, il est possible de s'appuyer sur le tableau 6.3 qui lie chaque scénario aux besoins exprimés par les météorologues. Ces besoins sont exprimés au chapitre 3.

On remarque que nous disposons de plusieurs scénarios pour chaque besoin, excepté pour le besoin "création de services composés". Aucun scénario ne concerne ce besoin puisqu'il ne porte pas précisément sur la méthode de composition automatique et adaptative.

On constate que les scénarios de choix et d'union sont identiques : les services Web utilisés et la requête de l'utilisateur sont identiques. Néanmoins, ils se distinguent dans le tableau par le fait que les résultats attendus sont différents. En effet, l'union de deux résultats de services Web normaux peut être complète alors que le choix sur ces deux mêmes services est toujours incomplet.

scénarios	prise en compte de l'hétérogénéité	reprise des pannes	complétion des données	amélioration de la qualité	estimation de la qualité	prise en compte de nouveaux services	création de services composés
sc_seq_A	•					•	
sc_seq_B	•					•	
sc_seq_C	•					•	
sc_choix_A		•	•	•	•	•	
sc_choix_B		•		•	•	•	
sc_choix_C		•		•	•	•	
sc_union_A			•	•	•	•	
sc_union_B			•	•	•	•	
sc_union_C			•	•	•	•	
sc_union1_A			•	•	•	•	
sc_union1_B			•	•	•	•	
sc_union1_C			•	•	•	•	
sc_union2_A			•	•	•	•	
sc_union2_B			•	•	•	•	
sc_union2_C			•	•	•	•	
sc_choix1_A		•		•	•	•	
sc_choix1_B		•		•	•	•	
sc_choix1_C		•		•	•	•	
sc_choix2_A		•		•	•	•	
sc_choix2_B		•		•	•	•	
sc_choix2_C		•		•	•	•	
sc_seq1_A	•					•	
sc_seq1_B	•					•	
sc_seq1_C	•					•	

TAB. 6.3 – Scénarios et besoins exprimés. Ce tableau compare les scénarios (1ère colonne) vis-à-vis des besoins exprimés en météorologie (1ère ligne). Chaque point représente le fait que le scénario vérifie un besoin exprimé (par exemple, le scénario de choix exclusif C vérifie que la méthode est capable de remplacer un service Web en panne). On remarque que la dernière colonne est vide : notre méthode d'évaluation ne vérifie pas que la méthode traite la création de services composés.

6.2.5.3 Evaluation de la qualité

L'analyse de la qualité s'effectue de la même façon que l'évaluation des fonctionnalités. J'établis un tableau synthétisant les résultats obtenus lors de l'exécution des scénarios. Ce tableau est formé de 4 colonnes, la première indiquant le nom du scénario et les suivantes étant :

- la seconde colonne indique les tests où le résultat obtenu est celui de meilleure qualité ;
- la troisième colonne indique le nombre de scénarios pour lesquels le résultat obtenu n'est pas celui de meilleure qualité ;
- la quatrième colonne récapitule le nombre de tests pour lesquels aucune information de qualité n'est disponible ou dont l'exécution a donné un résultat faux.

Contrairement à la méthode d'évaluation des fonctionnalités précédente, la meilleure qualité n'est pas déterminée dans le cas où les services Web sont parfaits, mais elle est déterminée par rapport à la meilleure qualité qui aurait dû être obtenue dans les conditions du scénario. Par exemple, si le scénario propose deux services de fonctionnalités identiques mais de qualité différente, la meilleure qualité obtenue pour ce scénario est celle du meilleur service Web. En revanche, pour un scénario ayant ces mêmes services Web mais où le service Web de meilleure qualité serait en panne, le résultat de meilleure qualité sera naturellement celui obtenu avec le service Web de moins bonne qualité.

De plus, le critère de qualité n'est pas fixe et celui-ci dépend de la méthode. Dans notre cas les méthodes utilisent un critère de qualité par service Web et réalisent le produit des qualités (cf chapitre 4 et 5) des services Web utilisés par la séquence pour déterminer la qualité du résultat. Si une autre méthode utilise un autre critère de qualité, celui-ci sera naturellement choisi. La qualité reste assez subjective, et donc notre atelier d'évaluation se contente d'évaluer si les méthodes prennent en compte un critère de qualité correctement, plutôt que d'évaluer une hypothétique qualité absolue.

Dans certains scénarios, les réponses de meilleure qualité sont les résultats sans réponse. Le tableau 6.4 est un exemple de tableau synthétisant les résultats pour l'évaluation de la qualité. Il est également le tableau de référence, c'est-à-dire le tableau qui aurait été obtenu par la méthode de composition idéale.

Par ailleurs, cette étape de l'évaluation ignore totalement la complètude du résultat. Une seule valeur avec la qualité la meilleure suffit à considérer que la méthode choisit bien la meilleure qualité. Seuls les résultats faux ou absents sont différenciés.

La comparaison du tableau synthétisant les résultats pour une méthode donnée avec le tableau de référence 6.4 permet de conclure quant à la capacité de la composition à prendre en compte la qualité à condition que le tableau des résultats soit identique au tableau de référence. Dans le cas contraire, il est difficile de se prononcer. Il est possible d'analyser dans quel cas la composition prend en compte la qualité et dans quel cas, elle ne la prend pas, en comparant scénario par scénario.

	1	2	3
sc_appel_A	1	0	0
sc_appel_B	1	0	0
sc_appel_C	0	0	1
sc_seq_A	1	0	0
sc_seq_B	1	0	0
sc_seq_C	0	0	1
sc_choix_A	1	0	0
sc_choix_B	1	0	0
sc_choix_C	1	0	0
sc_union_A	1	0	0
sc_union_B	1	0	0
sc_union_C	1	0	0

	1	2	3
sc_choix1_A	1	0	0
sc_choix1_B	1	0	0
sc_choix1_C	1	0	0
sc_choix2_A	1	0	0
sc_choix2_B	1	0	0
sc_choix2_C	1	0	0
sc_union1_A	1	0	0
sc_union1_B	1	0	0
sc_union1_C	1	0	0
sc_union2_A	1	0	0
sc_union2_B	1	0	0
sc_union2_C	1	0	0
sc_seq1_A	1	0	0
sc_seq1_B	1	0	0
sc_seq1_C	0	0	1

TAB. 6.4 – Tableau de référence pour un test réalisé avec une méthode idéale. La colonne 1 indique si le résultat du test est celui de meilleure qualité ; la colonne 2 rapporte si le résultat n'est pas celui de meilleure qualité ; la colonne 3 indique si le résultat comporte des données fausses ou absentes.

6.3 Implémentation

Cette section présente une implémentation de l'atelier d'évaluation, ainsi que les tests réalisés sur les trois prototypes de composition dynamique et adaptative du chapitre précédent. Pour réaliser ces tests, il faut définir de manière générale les services Web utilisés pour ces tests. Ensuite, je définis les requêtes ainsi que les résultats parfaits, qui sont les résultats de référence, afin d'obtenir le tableau synthétique pour l'évaluation des fonctionnalités. Enfin, je présente la description des services Web spécifiques à chacune des méthodes.

6.3.1 Choix des services Web pour l'implémentation d'un banc de test

Cette section définit les services Web qui rempliront ces scénarios. Plusieurs services Web peuvent convenir, mais l'ensemble de ces services Web doit répondre à certaines contraintes. Par exemple, les services Web S_b et S_c doivent pouvoir être appelés l'un après l'autre, le service Web S_c dépendant des données du services Web S_b ou bien les compositions S_k puis S_l doivent être équivalentes à la composition S_m puis S_n. Il faut donc choisir ces services Web en fonction de ces contraintes de manière à pouvoir remplir toutes les formes de composition. Par ailleurs, pour simplifier, on constate que certains services Web ont les mêmes contraintes, par exemple, S_b et S_c ont les mêmes contraintes que S_i et S_j. De cette façon, il est possible de regrouper les services Web de manière à minimiser le nombre de services Web à créer. Voici un regroupement possible :

$$S_a, S_b, S_d, S_f, S_i, S_k, S_p, S_r, S_x \quad (6.10)$$
$$S_c, S_j, S_l, S_q, S_s, S_v \quad (6.11)$$
$$S_e, S_g, S_m, S_t \quad (6.12)$$
$$S_n, S_u \quad (6.13)$$
$$S_h, S_o \quad (6.14)$$
$$S_v \quad (6.15)$$
$$S_w \quad (6.16)$$

Enfin, suivant les scénarios, ces services Web peuvent apparaître dans un des trois états *parfaits*, *normaux* ou en *panne*, comme nous pouvons le constater pour le service S_a.

Sept services Web différents, qui pourront être parfaits, normaux ou en panne, permettent d'effectuer une implémentation complète.

6.3.1.1 Les sept services Web en météorologie choisis

Je présente ici l'ensemble de services Web choisis de manière à correspondre aux besoins exprimés en météorologie. Bien sûr, il serait possible d'en choisir d'autres en fonction du domaine que l'on souhaite privilégier. Notre ensemble comporte les services Web suivants :

opérations $S_a, S_b, S_d, S_f, S_i, S_k, S_p, S_r, S_x$ (6.10) : NCEP fournissant des séries de températures en degrés Celsius ;

opérations $S_c, S_j, S_l, S_q, S_s, S_v$ (6.11) : conversion de températures de Celsius en Farenheit ;

opérations S_e, S_g, S_m, S_t (6.12) : NCEP fournissant des séries de températures en degrés Celsius ;

opérations S_n, S_u (6.13) : conversion de températures de Celsius en Farenheit ;

opérations S_h, S_o (6.14) : NCEP fournissant des séries de températures en degrés Farenheit ;

opération S_v (6.15) : fournit des coordonnées géographiques à partir d'un nom de ville et du nombre d'habitants. En entrée : (nom ville, nombre habitants), en sortie : (latitude, longitude).

opération S_w (6.16) fournit une période du calendrier grégorien à partir d'une période en nombre de seconde depuis 1970. En entrée : (période en nombre de seconde), en sortie : (période en calendrier grégorien).

On remarque que les services 6.10 et 6.12 proposent la même fonctionnalité, de même pour les services 6.11 et 6.13. Pourtant, ce sont des services Web différents et indépendants. En particulier, ces services Web, quand ils sont normaux, ne fournissent pas la même couverture spatio-temporelle. Ces services sont identiques lorsqu'ils sont parfaits. L'idéal pour l'outil d'évaluation est que chacun d'eux ait un recouvrement temporel ou spatial différent et complémentaire, de manière à fournir une partie des données demandées par la requête.

Dans notre cas, les services Web 6.10 et 6.12 fournissent des données complémentaires lorsqu'ils sont normaux :

- le service 6.10 fournit les données de tous les jours des années paires ;
- le service 6.12 fournit tous les jours des années impaires.

De plus, ces services Web fournissent des données dont la qualité est différente : le service Web 6.10 étant de meilleure qualité que le service 6.12 avec respectivement 0.9 et 0.4 en valeur de qualité.

6.3.2 Définition des requêtes et des résultats attendus pour chaque scénario

Une fois les services Web connus, nous devons définir les requêtes ainsi que les réponses attendues des méthodes.

Scénario `appel` :
- Requête : l'utilisateur souhaite obtenir une liste de couples `(date, température en degré Celsius)` à partir du triplet `(date range, latitude, longitude)`
- Résultats attendus : l'utilisateur obtient toutes les données souhaitées, c'est-à-dire tous les couples de dates et de températures en degré Celsius trouvées sur l'horizon temporel indiqué par l'intervalle de dates et au point indiqué par la latitude et la longitude

Scénario `seq` :
- Requête : l'utilisateur souhaite obtenir la liste de couples `(date, température en degré Farenheit)` à partir du triplet `(date range, latitude, longitude)`
- Résultats attendus : l'utilisateur obtient toutes les données souhaitées, c'est-à-dire tous les couples de date et température en degré Celsius entre les dates données et au point donné par la latitude et la longitude

Scénario `choix` :
- Requête : l'utilisateur souhaite obtenir la liste de tuples (date, température en degré Celsius) en fournissant le tuple (date range, latitude, longitude)
- Résultats attendus : l'utilisateur obtient les données du service 6.10

Scénario `union` :
- Requête : l'utilisateur souhaite obtenir la liste de tuples (date, température en degré Celsius) en fournissant le tuple (date range, latitude, longitude)
- Résultats attendus : l'utilisateur obtient les données du service 6.10

Scénario `choix1_A` :
- Requête : l'utilisateur souhaite obtenir la liste de tuples (date, température en degré Farenheit) en fournissant le tuple (date range, latitude, longitude)
- Résultats attendus : l'utilisateur obtient les données de la séquence de deux services 6.12 et 6.13

Scénario `choix2` :
- Requête : l'utilisateur souhaite obtenir la liste de tuples (date, température en degré Farenheit) en fournissant le tuple (date range, latitude, longitude)
- Résultats attendus : l'utilisateur obtient les données de meilleure qualité

Scénario `union1` :

- Requête : l'utilisateur souhaite obtenir la liste de tuples (date, température en degré Farenheit) en fournissant le tuple (date range, latitude, longitude)
- Résultats attendus : l'utilisateur obtient toutes les données souhaitées de meilleure qualité

Scénario `union2` :
- Requête : l'utilisateur souhaite obtenir la liste de tuples (date, température en Farenheit) en fournissant le tuple (date range, latitude, longitude)
- Résultats attendus : l'utilisateur obtient les données de meilleure qualité

Scénario `seq1` :
- Requête : l'utilisateur souhaite obtenir la liste de tuples (date, température en degré Celsius) en fournissant le tuple (date range, latitude, longitude)
- Résultats attendus : l'utilisateur obtient toutes les données souhaitées

6.3.3 Exemple des descriptions utilisées par les méthodes de composition

Comme la description des services Web est importante, je présente ici la méthode choisie pour décrire les services Web pour chacune des méthodes testées.

6.3.3.1 Descriptions utilisées par la méthode basée sur les graphes

La description, pour la méthode basée sur les graphes, du service Web fournissant les séries temporelles de températures en degrés Farenheit est la suivante :

```
service : getTemperatureFarenheit
location : http://localhost:8888/
input : ["geopoint", "period"]
output : ["temperatureFarenheit"]
```

Cette description comporte donc principalement les types d'entrée et les types de sortie. Le terme `temperatureFarenheit` représente une liste de températures en degré Farenheit. Chacune de ces entrées représente des états partiels. J'ai choisi de mettre dans le type l'unité utilisée afin de permettre à la méthode de distinguer les différents services de conversion d'unité. Les types utilisés sont les suivants :
- `geopoint` pour la localisation géographique ;
- `period` pour les intervalles de temps ;
- `temperatureFarenheit` pour les séries de température en Farenheit ;
- `temperatureCelsius` pour les températures en degré Celsius.

6.3.3.2 Descriptions utilisées par la méthode basée sur Prolog

La description, pour la méthode Prolog, du service Web fournissant les séries temporelles de températures en degré Farenheit est la suivante :

```
service : getTemperatureFarenheit
location : http://localhost:8888/
servicedesc :
service(service5, I, O) :-
```

```
find(geopoint(G1,G2), I),
find(period(B,E), I),
getseqtemperature(period(B,E), farenheit, geopoint(G1,G2), O)
```

Cette description comporte principalement un axiome Prolog qui relie les entrées et les sorties de chaque service Web, comme nous l'avons vu dans le chapitre 4.

6.3.3.3 Description utilisée par la méthode hybride

La description, pour la méthode hybride, du service Web fournissant les séries temporelles de températures en degrés Farenheit est définie par la localisation du service Web, *http ://localhost :8888/*, par la description des entrées et sorties utilisée par cette méthode :

- input : `["geopoint", "period"]`,
- output : `["temperature"]`,

et enfin par la description du comportement suivant :

func Service5 **exec**
    result = **{** sequence : **[ ] }** ;
    foreach i **in range**(period.begin, period.end) **do**
        tmp = **{ }** ;
        tmp.date = i ;
        tmp.geopoint = geopoint ;
        tmp.units = "Farenheit" ;
        result.sequence = **append**(result.sequence, get (tmp))
    done
return result

Dans notre cas, nous pouvons voir que nous n'avons pas conservé les unités dans la description des entrées et sorties, néanmoins nous les retrouvons dans la description du comportement. Les types utilisés pour la première description sont : `geopoint`, `period` et `temperature`.

6.3.4 Réalisation des tests avec les différents prototypes

Pour finir, j'ai réalisé les tests pour chacun des prototypes à l'aide de scripts. J'ai commencé par réaliser chacun des services Web nécessaires pour la réalisation de tous les scénarios en utilisant XML-RPC, comme je vais le montrer. Ensuite, j'ai créé un script qui, pour chaque scénario, lance les services Web nécessaires. De même, pour chaque scénario, j'ai réalisé des scripts exécutant les requêtes. Enfin un script réalise l'ensemble des scénarios pour un prototype donné. Chaque prototype étant différent, chaque script est différent. Les résultats sont enregistrés dans des fichiers que je compare manuellement aux résultats attendus.

6.3.4.1 XML-RPC pour les services Web

Les services Web utilisés ne suivent pas le protocole standard habituel qui est SOAP. La principale raison est d'ordre pratique : en effet il existe en Python une

librairie implémentant XML-RPC qui présente l'avantage d'être simple à mettre en œuvre.

XML-RPC est une spécification et un ensemble d'implémentation qui permettent aux programmes s'exécutant sur différentes combinaisons de systèmes d'exploitation, d'architectures et d'environnements de s'appeler entre eux à travers Internet [20, 114].

L'appel à un programme extérieur est réalisé grâce à une requête HTTP, de la même façon que peut le faire SOAP. La requête HTTP transporte un flux de données en XML. XML-RPC a été bâti de manière à être simple tout en permettant de transmettre, traiter et renvoyer des structures de données complexes [114].

Les types des données de base transmises à l'aide de XML-RPC sont :

- `<i4>` ou `<int>` qui sont des entiers de 4 octets,
- `<boolean>` qui sont soit vrais ou faux,
- `<string>` qui sont des chaînes de caractères
- `<double>` qui sont les nombres à virgule flottante à double précision,
- `<dateTime.iso8601>` qui sont les dates au format iso8601,
- `<base64>` qui sont les données binaires encodées en base 64.

A partir de ces types de base, XML-RPC définit des types complexes qui les combinent. Il définit les `<struct>` et les `<array>` qui sont respectivement des dictionnaires associatifs, qui associent un nom à une valeur et des listes de valeurs.

La spécification de XML-RPC est proche de celle de SOAP dans son but d'interopérabilité, et dans les technologies mises en œuvre comme le HTTP ou le XML. Ils se différencient, en particulier sur la gestion des espaces de nom, complètement absente dans XML-RPC et sur le fait que SOAP n'est pas spécifiquement défini pour être utilisé sur le protocole HTTP.

Chacun des services définis utilise un dictionnaire associatif pour représenter un message SOAP. Chacun de ces messages comporte les paramètres d'appel ou de retour du service Web. Bien que cette définition soit différente de celle de SOAP, l'approche est la même qu'avec les services Web standards : chaque service Web est défini à l'aide d'une signature typée. La réalisation de services Web utilisant des protocoles habituels tels que SOAP n'est pas simple techniquement mais ne présente aucune difficulté scientifique.

Pour illustrer, prenons l'exemple du service NCEP. Le service NCEP prend comme entrée un message comportant un *geopoint* et une *period* et renvoie un message comportant une liste appelée *temperatures*. La requête XML-RPC correspondant à une latitude de 48.86, une longitude de 2.33 et une période entre 2005-05-05 et 2005-05-15, est :

```
send: 'POST / HTTP/1.0
Host: localhost:8888
User-Agent: xmlrpclib.py/1.0.1 (by www.pythonware.com)
Content-Type: text/xml
Content-Length: 575

'send: "<?xml version='1.0'?>
<methodCall>
 <methodName>service1</methodName>
  <params>
   <param>
    <value><struct>
```

```
     <member>
      <name>geopoint</name>
      <value><struct>
       <member>
        <name>lat</name>
        <value><double>48.86</double></value>
       </member>
       <member>
        <name>lon</name>
        <value><double>2.33</double></value>
       </member>
      </struct></value>
     </member>
     <member>
      <name>period</name>
      <value><struct>
       <member>
        <name>begin</name>
        <value><int>20050505</int></value>
       </member>
       <member>
        <name>end</name>
        <value><int>20050515</int></value>
       </member>
      </struct></value>
     </member>
    </struct></value>
   </param>
  </params>
</methodCall>"
```

La réponse correspondant à l'appel de ce service Web est :

```
reply: 'HTTP/1.0 200 OK\r\n'
header: Server: BaseHTTP/0.3 Python/2.6.2
header: Date: Mon, 18 May 2009 15:09:52 GMT
header: Content-type: text/xml
header: Content-length: 6311
body: "<?xml version='1.0'?>
 <methodResponse>
  <params>
   <param>
    <value>
     <struct>
      <member>
       <name>temperatures</name>
       <value>
        <array><data>
         <value><struct>
          <member>
```

```
            <name>date</name>
             <value><int>20050505</int></value>
            </member>
            <member>
             <name>geopoint</name>
             <value><struct>
               <member>
                <name>lat</name>
                <value><double>48.86</double></value>
               </member>
               <member>
                <name>lon</name>
                <value><double>2.33</double></value>
               </member>
              </struct>
             </value>
            </member>
            <member>
             <name>quality</name>
             <value><double>0.90000000000000002</double></value>
            </member>
            <member>
             <name>unit</name>
             <value><string>Celsius</string></value>
            </member>
            <member>
             <name>value</name>
             <value><int>19</int></value>
            </member>
           </struct>
          </value>
          <value><struct>
           <member>
            <name>date</name>
            <value><int>20050506</int></value>
           </member>
           [.../...]
          </value>
          [.../...]
         </data>
        </array>
       </value>
      </member>
     </struct>
    </value>
   </param>
  </params>
</methodResponse>'
```

6.3.4.2 Exécution des scénarios

L'exécution des scénarios est réalisée principalement par trois scripts différents.

Le premier script a pour objectif de démarrer un serveur contenant les services Web. Ce script définit quels services Web doivent être démarrés et quels services Web doivent être arrêtés, ainsi que leur statut : normal ou parfait. Les services Web normaux sont réalisés à partir des services Web parfaits. Le service Web parfait fournissant toutes les données, je retire la moitié des données qu'il a fournies pour réaliser le service Web normal correspondant. De plus je ne retire pas cette moitié au hasard, je la retire de façon à ce qu'au moins une valeur puisse être complétée par un autre service Web normal dans chaque scénario où c'est nécessaire. Le serveur reste en route tout au long du déroulement d'un scénario puis il est arrêté une fois le scénario terminé. Les scénarios sont exécutés sur une même machine mais ils ne peuvent pas être exécutés en même temps.

Le deuxième script se charge d'exécuter la requête de l'utilisateur, afin d'obtenir le résultat de la composition. Ces scripts sont spécifiques à chacune des méthodes, mais suivent le même schéma d'initialisation et d'exécution. D'abord le script définit les données connues par la méthode au moment de la requête, puis la méthode est exécutée avec les paramètres et la requête de l'utilisateur.

Le dernier script a pour but de réaliser tous les scénarios, les uns après les autres. Il commence par démarrer le serveur du premier scénario, puis exécute la requête. Ensuite il enregistre le résultat dans un fichier qui servira pour l'analyse. Enfin, il arrête le serveur pour démarrer le serveur du scénario suivant.

6.4 Résultats et Analyse de l'outil

Dans cette section, je présente les résultats obtenus par les trois prototypes. Je présente également une analyse de l'outil d'évaluation lui-même. La première partie analyse les fonctionnalités des méthodes de composition, la seconde analyse la gestion de la qualité de ces prototypes. Dans la dernière partie j'analyse l'outil d'évaluation lui-même.

6.4.1 Fonctionnalités des prototypes

Un premier résultat probant du test des fonctionnalités est que chacune des trois méthodes a pu réaliser les tests et qu'aucun de ces tests n'a fourni de réponses fausses. Les trois prototypes sont donc capable de générer dynamiquement des compositions en fonction de l'environnement et des connaissances dont elles disposent.

Les résultats obtenus pour les tests des fonctionnalités sont reportés dans les tableaux 6.5 et 6.6. Ces résultats montrent que toutes les méthodes testées réalisent l'appel à un simple service Web, ainsi que la séquence de services Web de manière automatique et adaptative. Ceci représente une avancée par rapport à la méthode employée par SoDa qui ne prend pas en compte de nouveaux services Web.

Par ailleurs, les méthodes basées sur les graphes ou sur Prolog réalisent également le choix d'une composition, chacune d'elles remplaçant une composition défaillante par une autre qui fonctionne.

Néanmoins, ces deux méthodes ne réalisent pas l'union Ce résultat était tout de même prévisible compte tenu de leur nature, chacune d'elles ne disposant pas de moyen efficace pour vérifier le résultat d'une composition.

En ce qui concerne la méthode hybride, les résultats sont différents puisque, contrairement aux deux autres méthodes, elle ne réalise pas le choix mais elle réalise l'union. Cette méthode présente donc une amélioration par rapport aux deux autres méthodes.

Enfin, les tests réalisés pour les deux premières méthodes, c'est-à-dire la méthode des graphes et la méthode basée sur Prolog, sont conformes aux premières analyses correspondantes du chapitre 2 : la méthode des graphes correspondant à une méthode de planification et la méthode basée sur Prolog correspond à de la preuve de programme.

De plus, les résultats de ces tests sont également conformes aux résultats trouvés dans la littérature, comme nous l'avons vu dans le chapitre 2 sur l'état de l'art. Ces deux premières analyses me confortent dans l'idée que cet outil d'évaluation semble raisonnable, même s'il reste impossible de vérifier de façon certaine que tous les besoins en météorologie seront remplis. En revanche, cet outil montre ses limites en ne permettant pas de mettre en évidence certains défauts de ces méthodes, comme la limitation du nombre d'utilisation d'un même service dans une composition ou bien la limitation du nombre de services Web utilisables par la méthode basée sur Prolog. Ces limitation sont développées plus loin.

	Méthode des graphes				
	1	**2**	**3**	**4**	
appel_A	1	0	0	0	OK
appel_B	0	1	0	0	OK
appel_C	0	0	1	0	OK
seq_A	1	0	0	0	OK
seq_B	0	1	0	0	OK
seq_C	0	0	1	0	OK
choix_A	1	0	0	0	OK
choix_B	0	1	0	0	OK
choix_C	0	1	0	0	OK
union_A	1	0	0	0	OK
union_B	0	1	0	0	NO
union_C	0	1	0	0	NO
choix1_A	1	0	0	0	OK
choix1_B	0	1	0	0	OK
choix1_C	0	1	0	0	OK
choix2_A	1	0	0	0	OK
choix2_B	0	1	0	0	OK
choix2_C	0	1	0	0	NO
union1_A	1	0	0	0	OK
union1_B	0	1	0	0	NO
union1_C	0	1	0	0	NO
union2_A	1	0	0	0	OK
union2_B	0	1	0	0	NO
union2_C	0	1	0	0	NO
seq1_A	0	0	1	0	NO
seq1_B	0	0	1	0	NO
seq1_C	0	0	1	0	NO

	Méthode Prolog				
	1	**2**	**3**	**4**	
appel_A	1	0	0	0	OK
appel_B	0	1	0	0	OK
appel_C	0	0	1	0	OK
seq_A	1	0	0	0	OK
seq_B	0	1	0	0	OK
seq_C	0	0	1	0	OK
choix_A	1	0	0	0	OK
choix_B	0	1	0	0	OK
choix_C	0	1	0	0	OK
union_A	1	0	0	0	OK
union_B	0	1	0	0	NO
union_C	0	1	0	0	NO
choix1_A	1	0	0	0	OK
choix1_B	0	1	0	0	OK
choix1_C	0	1	0	0	OK
choix2_A	1	0	0	0	OK
choix2_B	0	1	0	0	OK
choix2_C	0	1	0	0	NO
union1_A	1	0	0	0	OK
union1_B	0	1	0	0	NO
union1_C	0	1	0	0	NO
union2_A	1	0	0	0	OK
union2_B	0	1	0	0	NO
union2_C	0	1	0	0	NO
seq1_A	0	0	1	0	NO
seq1_B	0	0	1	0	NO
seq1_C	0	0	1	0	NO

TAB. 6.5 – Evaluation des scénarios pour les prototypes basés sur les graphes et sur Prolog. Un seul test a été réalisé. La colonne 1 indique si le test est réussi ; la colonne 2 rapporte si le test est partiellement réussi (il manque au moins une donnée) ; la colonne 3 indique si le test a échoué (aucune donnée n'a été fournie) ; la colonne 4 indique si le résultat du test est faux (contenant des données fausses).

	Méthode hybride				
	1	**2**	**3**	**4**	
appel_A	1	0	0	0	OK
appel_B	0	1	0	0	OK
appel_C	0	0	1	0	OK
seq_A	1	0	0	0	OK
seq_B	0	1	0	0	OK
seq_C	0	0	1	0	OK
choix_A	1	0	0	0	OK
choix_B	1	0	0	0	OK
choix_C	0	1	0	0	OK
union_A	1	0	0	0	OK
union_B	1	0	0	0	OK
union_C	0	1	0	0	OK
choix1_A	1	0	0	0	OK
choix1_B	1	0	0	0	OK
choix1_C	0	1	0	0	OK
choix2_A	1	0	0	0	OK
choix2_B	1	0	0	0	OK
choix2_C	1	0	0	0	OK
union1_A	1	0	0	0	OK
union1_B	1	0	0	0	OK
union1_C	0	1	0	0	OK
union2_A	1	0	0	0	OK
union2_B	1	0	0	0	OK
union2_C	1	0	0	0	OK
seq1_A	0	0	1	0	NO
seq1_B	0	0	1	0	NO
seq1_C	0	0	1	0	NO

TAB. 6.6 – Evaluation des scénarios pour le prototype hybride. Un seul test a été réalisé. La colonne 1 indique si le test est réussi ; la colonne 2 rapporte si le test est partiellement réussi (il manque au moins une donnée) ; la colonne 3 indique si le test a échoué (aucune donnée n'a été fournie) ; la colonne 4 indique si le résultat du test est faux (contenant des données fausses).

6.4.2 Tests de gestion de la qualité

Les résultats des tests sur la gestion de la qualité sont reportés dans les tableaux 6.7, 6.8 et 6.9. Le résultat des tests de la qualité montre simplement que toutes les méthodes de composition qui trouvent un résultat prennent en compte la qualité. Il n'y a rien d'étonnant à cela, compte-tenu qu'elles utilisent toutes une méthode similaire pour la prise en compte de la qualité. Il faut tout de même remarquer que les résultats obtenus par la méthode hybride comportent uniquement des données de la meilleure qualité disponible dans chaque scénario, contrairement aux deux autres. Ceci vient de l'implémentation de l'union dans la méthode hybride qui complète les résultats. Néanmoins, en contre-partie, elle mélange des données de différentes qualités, ce qui n'est pas forcément souhaitable, par exemple, si un utilisateur désire des données de qualité homogène.

	1	**2**	**3**	
sc_appel_A	1	0	0	OK
sc_appel_B	1	0	0	OK
sc_appel_C	0	0	1	OK
sc_seq_A	1	0	0	OK
sc_seq_B	1	0	0	OK
sc_seq_C	0	0	1	OK
sc_choix_A	1	0	0	OK
sc_choix_B	1	0	0	OK
sc_choix_C	1	0	0	OK
sc_union_A	1	0	0	OK
sc_union_B	1	0	0	OK
sc_union_C	1	0	0	OK

	1	**2**	**3**	
sc_choix1_A	1	0	0	OK
sc_choix1_B	1	0	0	OK
sc_choix1_C	1	0	0	OK
sc_choix2_A	1	0	0	OK
sc_choix2_B	1	0	0	OK
sc_choix2_C	1	0	0	OK
sc_union1_A	1	0	0	OK
sc_union1_B	1	0	0	OK
sc_union1_C	1	0	0	OK
sc_union2_A	1	0	0	OK
sc_union2_B	1	0	0	OK
sc_union2_C	1	0	0	OK
sc_seq1_A	0	0	1	NO
sc_seq1_B	0	0	1	NO
sc_seq1_C	0	0	1	NO

TAB. 6.7 – Tableau de synthèse de la qualité pour la méthode des graphes. La colonne 1 indique si le résultat du test est de la meilleure qualité ; la colonne 2 rapporte si le résultat n'est pas de la meilleure qualité ; la colonne 3 indique si le résultat comporte des données fausses ou absentes.

	1	2	3	
sc_appel_A	1	0	0	OK
sc_appel_B	1	0	0	OK
sc_appel_C	0	0	1	OK
sc_seq_A	1	0	0	OK
sc_seq_B	1	0	0	OK
sc_seq_C	0	0	1	OK
sc_choix_A	1	0	0	NO
sc_choix_B	1	0	0	NO
sc_choix_C	1	0	0	NO
sc_union_A	1	0	0	NO
sc_union_B	1	0	0	NO
sc_union_C	1	0	0	NO

	1	2	3	
sc_choix1_A	1	0	0	OK
sc_choix1_B	1	0	0	OK
sc_choix1_C	1	0	0	OK
sc_choix2_A	1	0	0	OK
sc_choix2_B	1	0	0	OK
sc_choix2_C	1	0	0	OK
sc_union1_A	1	0	0	OK
sc_union1_B	1	0	0	OK
sc_union1_C	1	0	0	OK
sc_union2_A	1	0	0	OK
sc_union2_B	1	0	0	OK
sc_union2_C	1	0	0	OK
sc_seq1_A	0	0	1	NO
sc_seq1_B	0	0	1	NO
sc_seq1_C	0	0	1	NO

TAB. 6.8 – Tableau de synthèse de la qualité pour la méthode sur Prolog. La colonne 1 indique si le résultat du test est de la meilleure qualité ; la colonne 2 rapporte si le résultat n'est pas de la meilleure qualité ; la colonne 3 indique si le résultat comporte des données fausses ou absentes.

	1	2	3	
sc_appel_A	1	0	0	OK
sc_appel_B	1	0	0	OK
sc_appel_C	0	0	1	OK
sc_seq_A	1	0	0	OK
sc_seq_B	1	0	0	OK
sc_seq_C	0	0	1	OK
sc_choix_A	1	0	0	NO
sc_choix_B	1	0	0	NO
sc_choix_C	1	0	0	NO
sc_union_A	1	0	0	NO
sc_union_B	1	0	0	NO
sc_union_C	1	0	0	NO

	1	2	3	
sc_choix1_A	1	0	0	OK
sc_choix1_B	1	0	0	NO
sc_choix1_C	1	0	0	NO
sc_choix2_A	1	0	0	NO
sc_choix2_B	1	0	0	NO
sc_choix2_C	1	0	0	NO
sc_union1_A	1	0	0	NO
sc_union1_B	1	0	0	NO
sc_union1_C	1	0	0	NO
sc_union2_A	1	0	0	NO
sc_union2_B	1	0	0	NO
sc_union2_C	1	0	0	NO
sc_seq1_A	0	0	1	NO
sc_seq1_B	0	0	1	NO
sc_seq1_C	0	0	1	NO

TAB. 6.9 – Tableau de synthèse de la qualité pour la méthode hybride. La colonne 1 indique si le résultat du test est de la meilleure qualité ; la colonne 2 rapporte si le résultat n'est pas de la meilleure qualité ; la colonne 3 indique si le résultat comporte des données fausses ou absentes.

6.4.3 Analyse de l'outil

Cet outil d'évaluation fournit des scénarios qui sont spécifiques à mon domaine d'étude. Cependant, il reste général parce qu'il laisse la possibilité de choisir les services Web. Par ailleurs, j'ai choisi de ne pas inclure dans cet outil de moyens d'évaluer les performances en terme de vitesse de réponse, et ce, parce que ces performances ne sont pas encore exigées par les utilisateurs de services Web en météorologie. Je considère que la rapidité de choix de composition fait partie de l'optimisation, et, à l'heure actuelle, le problème le plus important reste de réaliser une composition sûre. Le problème de la performance sera probablement important dans l'avenir, compte-tenu du nombre en expansion des outils de description des services Web et du nombre grandissant de ces services.

Cet outil prend en compte les exigences de qualité des données en météorologie, en différenciant le choix de l'union. En revanche, l'outil ne donne pas de critère précis de qualité. Les critères de qualité sont variables : parfois ils portent sur l'incertitude, parfois ils portent sur la complétude. Par exemple, certains utilisateurs souhaitent des séries temporelles de données d'incertitude homogène alors que d'autres préfèrent avoir les données avec les incertitudes les plus faibles. Pour ces raisons, cet outil laisse la liberté de choix des critères de qualité.

Pour utiliser cet outil d'évaluation, il faut définir les services Web remplissant les différentes formes de scénarios. J'en ai défini 7 à titre d'exemple. L'utilisateur de l'outil est libre de les changer à condition que l'ensemble des nouveaux services Web remplisse toujours toutes les formes de composition. Selon moi, le fait de changer ces services Web ne devrait pas changer les résultats du test ; la seule variable ayant des conséquences importantes est la description des services Web. En effet, lors de la réalisation de l'outil j'ai effectué des tests avec le prototype basé sur les graphes avec un type unique "temperature". La conséquence de cette description très abstraite a été des résultats régulièrement faux, car la méthode utilisait indifféremment le service Web donnant ses résultats en degrés Celsius et celui donnant des degrés Farenheit.

Cet outil d'évaluation n'est pas parfait et il présente encore des faiblesses. Sa principale faiblesse est de ne pas évaluer le comportement des méthodes qui se modifie au cours du temps en présence d'un contexte Internet qui change. On peut imaginer des solutions capables de prendre en compte les différents contextes Internet réels rencontrés pour établir plus justement un plan, par exemple une méthode dotée d'une capacité d'apprentissage, mais ce caractère ne sera pas identifié par l'outil actuel. On remarquera que cet outil se concentre sur l'évaluation de la capacité à composer de façon automatique et adaptative, mais pas sur la découverte des services Web, car ce problème est en dehors du cadre de cette thèse.

Par ailleurs, l'outil a également montré ses limites dans l'évaluation des trois méthodes présentées. En effet, il n'a pas mis en évidence le fait que la méthode basée sur les graphes et la méthode hybride ne sont pas capables d'utiliser plus de deux fois un même service Web dans une même composition. Il n'a pas montré non plus que la méthode basée sur Prolog ne peut pas utiliser plus qu'un certain nombre de services Web. Néanmoins, cette limite reste acceptable compte-tenu que ces défauts sont inhérents aux méthodes.

Cet outil d'évaluation des méthodes de composition possède des limites. Néanmoins, il est capable de valider à peu près toutes les formes de composition possibles, du simple choix aux unions les plus complexes et il recouvre bien tous les besoins de la météorologie comme le montre le tableau 6.3 excepté donc pour la création de

nouveaux services Web. Cet outil d'évaluation a été conçu pour évaluer les différentes méthodes de composition adaptative vis-à-vis des améliorations attendues en météorologie mais il peut être étendu facilement à des domaines proches, dont les services Web ont des propriétés similaires (comme par exemple la propriété d'être sans état) et fournissent des données représentatives de phénomènes physiques ayant des relations logiques entre elles.

Chapitre 7

Conclusion

L'objectif de cette thèse était de proposer une méthode de composition automatique et adaptative des services Web appliquée au domaine de la météorologie. Une composition automatique est une composition ne nécessitant pas l'intervention humaine lors de la composition. Une composition adaptative est une composition qui prend en compte le contexte internet pour réaliser la composition, afin, par exemple, de détecter et de remplacer les services Web en panne par d'autres équivalents. Les méthodes de composition actuelles ne sont pas satisfaisantes, car le résultat peut être incertain ou faux.

Dans le cas de la météorologie, les services Web présentent des caractéristiques qui peuvent être exploitées. Les services Web en météorologie sont de deux natures différentes, d'abord les algorithmes et ensuite les services Web fournissant des données. Ces services Web peuvent être vus comme des services Web sans-état, car ils ne conservent pas d'information liée aux entrées qui leur ont été fournies. Cette caractéristique simplifie la composition et offre un contexte favorable à la composition automatique et adaptative.

La composition de services Web dans le domaine de la météorologie a été définie dans le chapitre 3 et met en évidence les formes des compositions souhaitées par les météorologues. Elle montre également que la définition de l'union du résultat des services Web est un point important en météorologie. Cette formalisation m'a permis de réaliser un prototype exploitant les caractéristiques des services Web en météorologie. Cette composition propose une méthode pour l'union des résultats d'exécution des services Web.

Mon étude bibliographique m'a permis de constater que dans le domaine de la composition de services Web, les méthodes proposées étaient évaluées de façon subjective, leur évaluation pointant simplement ici et là leurs faiblesses et leurs qualités sans aucune exhaustivité ni méthode. J'ai donc, dans un premier temps, conçu un outil d'évaluation de la pertinence des compositions de services Web pour le domaine de la météorologie. Afin de définir de manière précise et pertinente cet outil, j'ai commencé par recueillir les besoins, en matière de composition, auprès de spécialistes en météorologie. J'ai ensuite formalisé ces besoins sous la forme d'un ensemble représentatif de scénarios dont l'objectif est de mettre, les méthodes de composition à tester, dans des situations à la fois représentatives et réelles et de manière à ce qu'elles respectent les besoins exprimés par les utilisateurs en météorologie. Cet outil a permis de mettre en évidence des qualités ainsi que des défauts des méthodes de composition ; il a aussi

permis d'identifier des limitations dans son utilisation.

L'analyse des problèmes liés aux différentes approches des méthodes de composition existantes nous a alors permis de proposer une nouvelle méthode de composition, dite méthode hybride. Cette nouvelle méthode bénéficie des propriétés des méthodes existantes étudiées et s'appuie sur une méthode de description émergente : OWL-S. La méthode de composition proposée contribue à montrer l'intérêt des méthodes de description de type OWL-S. Réciproquement, ce type de méthode de description montre que mon approche pour la composition de services Web est raisonnable. En revanche, ma méthode invente une interprétation nouvelle des descriptions en définissant la notion de fonction de base, qui n'est pas explicitement proposée par OWL-S. On remarquera que les deux types de description que j'utilise sont deux abstractions différentes d'un même service Web, elles sont donc liées. Une mauvaise utilisation du langage peut conduire à des résultats incertains, en particulier, si ces deux descriptions se contredisent.

J'ai ensuite appliqué l'outil d'évaluation de la pertinence des méthodes de composition à la nouvelle méthode que je propose. Les résultats issus de ces tests montrent que l'objectif de ma thèse, c'est-à-dire répondre au mieux aux besoins en météorologie, sont partiellement atteints. L'analyse des résultats montre que la méthode hybride prend en compte la qualité de façon globale et donc réalise le choix des compositions de meilleure qualité entre compositions. Cette méthode ne permet pas de définir de critère complexe de qualité de données.

Je propose ensuite plusieurs pistes pour améliorer la méthode hybride.

Une première piste est l'amélioration du calcul de la qualité : étant donné que chaque composition est décrite par son comportement, il est possible d'imaginer des formules de calcul complexe de la qualité, prenant en compte les opérations réalisées lors de la composition.

Une deuxième piste concerne le langage utilisé dans le prototype. Bien que ce langage soit suffisant, il peut être amélioré de façon à faciliter son utilisation, par exemple en mettant complètement en œuvre OWL-S.

Une troisième piste concerne la terminaison des programmes. Le langage actuel garantit cette terminaison grâce à l'interdiction de l'appel récursif aux fonctions et à la limitation des boucles sur des listes finie. Cette limite pourrait être levée et être remplacée par une limite de temps d'exécution ou en nombre d'opérations. Il faudrait analyser l'impact du changement de cette limite.

Une autre piste d'amélioration concerne la définition d'opérateurs mathématiques ayant des propriétés prédéfinies et pouvant être normalisés et donc permettre la comparaison de formules entre elles. De plus, dans la ligne de ce type d'amélioration, il serait judicieux de permettre de définir des propriétés sur les fonctions de base, comme des notions d'équivalence entre fonctions de base, permettant de définir la permutabilité, ou d'autres propriétés plus complexes. Néanmoins, l'utilisation des propriétés d'équivalence introduit souvent des espaces de recherche infinis.

La planification ou les méthodes basées sur la preuve de programmes ne sont pas satisfaisantes car leurs espaces de recherche sont, dans le cas habituel, infinis. Ces méthodes, se concentrent habituellement sur les données et leurs transformations. Néanmoins, si on tient compte des caractéristiques d'Internet où, pour répondre à un utilisateur, seul un nombre limité de services Web peut être appelé, le nombre de compositions possibles devient fini. En appliquant cette idée à la méthode hybride, le générateur de composition devient un générateur de formules n'utilisant que les compositions autorisées par le langage comportant un nombre limité d'appels de services

Web ; le nombre de formules dans ce cas est ainsi limité.

La composition automatique et adaptative de services Web reste une technologie d'avant-garde, malgré l'effort de recherche effectué ces dernières années. La composition de services Web promet un meilleur accès aux données disponibles et une amélioration de la qualité des données. Néanmoins, un long chemin reste à parcourir, en particulier dans les standards actuels. Ces derniers sont encore sous-utilisés et sont encore très flous pour la plupart des utilisateurs potentiels. De plus, ils restent difficiles à utiliser.

Par ailleurs, une des difficultés rencontrées est l'inter-compréhension des systèmes d'information. Au-delà de l'interopérabilité, qui consiste à pouvoir communiquer entre les entités informatiques, il est nécessaire de mettre en place des taxinomies, ontologies et thésaurus communs, ces derniers donnant un sens à Internet. Contrairement à la tendance actuelle qui consiste à être le plus consensuel possible en choisissant des définitions multiples pour la même chose ou des définitions tellement générales qu'il devient impossible de différencier des données entres elles, ces définitions gagneraient à être précises et claires. Dans l'avenir, une attention particulière sera portée sur ces descriptions qui sont l'essence des méthodes de composition.

La composition automatique de services Web sera réalisable à partir du moment où la description des services Web sera, dans les faits, standard. J'ai d'ailleurs travaillé en ce sens en proposant un thésaurus (cf. Annexe A) pour les observations du rayonnement solaire, utilisé dans les échanges actuels autour du service SoDa de demain.

Annexe A

Thésaurus

Gschwind Benoît, Lionel Ménard, Thierry Ranchin, Lucien Wald, Paul Stackhouse, 2007. A proposal for a thesaurus for web services in solar radiation. In Proceedings EnviroInfo 2007, O. Hryniewicz, J. Studzinski and M. Romaniuk (Eds), Shaker Verlag, vol. 1, pp. 135-142.

A Proposal for a Thesaurus for Web Services in Solar Radiation

Benoît Gschwind [1], Lionel Ménard[1], Thierry Ranchin[1], Lucien Wald[1] and Paul Stackhouse [2]

Abstract

Metadata are necessary to discover, describe and exchange any type of information, resource and service at a large scale. A significant amount of effort has been made in the field of geography and environment to establish standards. Efforts still remain to address more specific domains such as renewable energies. This communication focuses on solar energy and more specifically on aspects in solar radiation that relate to geography and meteorology. A thesaurus in solar radiation is proposed for the keys elements in solar radiation namely time, space and radiation types. The importance of time-series in solar radiation is outlined and attributes of the key elements are discussed. An XML schema for encoding metadata is proposed. The exploitation of such a schema in web services is discussed. This proposal is a first attempt at establishing a thesaurus for describing data and applications in solar radiation.

1. Introduction

Solar radiation is a domain where data are rare. The few providers are usually the National Meteorological Services which are geographically distributed. Such data are necessary to a large number of users (companies, consultants, researchers, students) in various business domains (energy, building engineering, health, agriculture, meteorology, climate…). Consequently, many exchanges of data occur. A recent survey conducted by the International Energy Agency (IEA) emphasises the heterogeneity observed in the data in various aspects (Cros et al. 2004). Access to data is complicated by the various types of data, various storage standards, various units, various time systems (coordinated universal time, mean solar time, true solar time, local time), diversity in information properties stored in databases: sampling support (e.g., pixel size or pin-point measurement), observational period, frequency of individual observations and averaging time intervals, etc. The survey identifies three major problems to solve in order to supply customers with information relevant to their requests: improved access to information, improved description and knowledge in space and time of the radiation field and related quantities, and improved matching to actual needs. It suggests that combining "measured or assessed data and physical models permits convenient access to the information. Co-operation between several sources of information provides fruitful results. Efforts made in integrated information systems and co-operative systems have to be supported to overcome the technical limits of measurements by using the information and communication technologies" (Cros et al. 2004).

We believe that an appropriate exploitation of metadata and web services within a collaborative system is an efficient means to solve these problems and to bring knowledge, applications and data when and where it is needed. The SoDa Service is such a collaborative information system and has been created to that aim. The success in exploitation since 2003 demonstrates that i) as foreseen by the IEA survey there is a real interest in improving access, ii) a collaborative system is an efficient solution, iii) the use of metadata permits to homogenise the graphical user interface, inputs from and outputs to users. The thesaurus of the SoDa Service is specific. An increase of capabilities in dissemination requires on the one hand, the adoption of standards and on the other hand, an extension of this thesaurus to better describe

[1] Ecole des Mines de Paris / Armines, BP 207, 06904 Sophia Antipolis cedex, France. Email: benoit.gschwind@ensmp.fr
[2] NASA Langley Research Center, Hampton, VA, USA

Gschwind Benoît, Lionel Ménard, Thierry Ranchin, Lucien Wald, Paul Stackhouse, 2007. A proposal for a thesaurus for web services in solar radiation. In Proceedings EnviroInfo 2007, O. Hryniewicz, J. Studzinski and M. Romaniuk (Eds), Shaker Verlag, vol. 1, pp. 135-142.

specific terms in solar radiation. The projects IEA SHC #36 (2006) and EC-funded MESoR (project FP7 #038655 "management and exploitation of solar resource knowledge") partly devoted to that goal.

Metadata are necessary to ease data exchange at a large scale. A large amount of effort has been made in the field of geography and environment to establish standards. Efforts still remain to address more specific domains. This communication focuses on solar energy and more specifically on aspects in solar radiation that relate to geography and meteorology. It is a first attempt at establishing a thesaurus for describing data and applications in solar energy. A XML schema is proposed with focus on the exploitation of such a thesaurus in web services. The proposal exploits the standards available in time (ISO, W3C), geographic information (ISO, INSPIRE, GML) as well as the outcomes of several working groups for metadata in meteorology and related domains.

Users points of view are taken into account to draft this thesaurus. These views have been expressed when designing commercial products (ESRA, Meteonorm), during projects in USA, Canada or EU where dissemination is an important issue (NASA-SSE, RETScreen, EU-Satellight, EU-SoDa, EU-MESoR) and works of the IEA and the Group on Earth Observation (GEO). In addition, surveys are periodically made by managers of information systems such as NASA-SSE, RETScreen and SoDa Service.

There are three different levels that metadata may be used for: "discovery metadata", "exploration metadata" and "exploitation metadata". Each of these purposes requires different levels of information. Users needs, low number of available services and attitude of the service providers show that presently, the emphasis should be put on "exploitation metadata". These metadata describe those properties required to access, transfer, load, interpret and apply the data in the end application where it is exploited. This class of metadata often includes the details of a data dictionary, the data organisation, projection and geometric characteristics. The communication focuses on this level of metadata.

A major concern in our development and operation of the collaborative information system is the intellectual property rights (IPR). The attitude of the projects IEA SHC #36 and MESoR regarding these aspects is a respect of the IPR of the service provider with no obligation towards the collaborative information system, except those resulting from the commercial agreements if any. The "exploitation metadata" should include these aspects. However, this communication does not deal with details in IPR. The main reason is that we do not have enough experience to be able to elaborate efficienctly details on IPR with service providers and customers. For example, no agreement was reached on payment schemes in a recent workshop gathering customers and service providers (EMP 2007). Metadata proposed by ISO 19115 in sections: Identification, Distribution, Citation and Responsible Party, are believed to cover our needs in a first phase. If necessary, extension may be possible for composition of services in the same way than discussed in this communication. The encoding process of these metadata may follow ISO 19118.

2. Thesaurus in solar radiation

Gathering these standards, requirements and know-how gained in managing information systems, we propose a thesaurus for the main elements in solar radiation. Several thesauri exist in meteorology (AMS 2007; FAO 2007; UNESCO 2007; WMO 1966). None of them describes solar radiation accurately enough. Consequently, we establish a thesaurus that is specific to solar radiation. It comprises several elements, some of them relating to time and others to radiation. This thesaurus is based on the works of CIE (1970), Envisolar (2004) and Wald (2007).

Time is a very important matter in solar radiation. The daily rotation of the earth about itself determines a day which is divided in 24 h as an average. The time defined in this way is called the mean solar time (MST). The sun is approximately at zenith when the MST is equal to 12 h. However, because the orbit of the earth is an ellipse, the sun does not reach its highest position in the sky at 12 h MST every day. The highest position is called solar noon and is reached every day at 12 h in true solar time (TST). The difference between MST and TST differs each day and may reach up to 17 min. The time standard is

Gschwind Benoît, Lionel Ménard, Thierry Ranchin, Lucien Wald, Paul Stackhouse, 2007. A proposal for a thesaurus for web services in solar radiation. In Proceedings EnviroInfo 2007, O. Hryniewicz, J. Studzinski and M. Romaniuk (Eds), Shaker Verlag, vol. 1, pp. 135-142.

Coordinated Universal Time (UTC) and is the basis of the legal time. It differs by less than 1 s per day from the so-called Universal Time (UT), which is equal to the MST for longitude 0. Finally, legal time is the time legally used in one country. Several countries make use of daylight saving time, that adds 1 h to the winter legal time (summer in South hemisphere). All these times are used in solar radiation. Thus, the first elements of the thesaurus are:

- **Legal time, local time**: The time used legally in a given country;
- **Mean solar time**: The time determined locally by dividing the average duration of a rotation by 24 h. Mean solar time is equal to 12 h when the sun is at its zenith as an annual average;
- **True solar time**: The time for which the sun is actually at its zenith when it is 12 h. This depends on the day of the year and longitude of the site;
- **Universal Time** (UT): The mean solar time for the longitude 0.

The amount of power received by a surface depends on its geographical location. Thus, space is an important matter. This is well described in several thesauri and there is no need to adress it in a specific way in solar radiation.

Two radiation types are used to describe radiation. **Irradiance** is defined as a power received per area or expressed differently as radiant flux of any origin incident onto an area. **Irradiation** is the energy received per area; it is an irradiance integrated over a certain period of time.

The radiation at ground, whether it is expressed as irradiance or irradiation, consists of several components: direct, diffuse and reflected. This decomposition may be of importance for several solar energy conversion systems. For convenience, one may add to these components the DNI (direct normal irradiance, or irradiation). The definitions are the following:

- **Diffuse** component: The downward scattered shortwave radiation coming from the whole hemisphere, with the exception of the solid angle of the sun disc and of the part of the radiation reflected by the ground in the case of an inclined receiving plane;
- **Direct** radiation: The shortwave radiation coming from the solid angle of the sun disc;
- **Reflected** radiation: The radiation reflected by the ground and received on an inclined plane;
- **Global** radiation: The shortwave radiation received at ground level; it is the sum of the direct, diffuse and reflected radiations;
- Direct normal radiation (**DNI**): The direct radiation received by a plane normal to the direction of the solar rays.

The spectral distribution of the radiation, i.e. the distribution of radiation as a function of the wavelength, varies in time and space. Its knowledge is requested for a better knowledge of the daylight, or UV radiation or any photo-energy system. Thus, we can define **spectral irradiance**, respectively **spectral irradiation**, as the spectral distribution of the irradiance, respectively irradiation, or the irradiance, respectively irradiation, integrated over a spectral window. This spectral property can be combined with the component, e.g., the spectral direct irradiance. If the spectral property is not mentioned, then it is understood that the radiation is considered as being the total, or broadband, radiation. This corresponds to an integration over the whole spectrum.

3. Description and attributes

How important are time-series in our activity should be underlined. Most of the information exchanged is presently under this form. We choose to focus on time-series and to use them as a basis for the description of information. A time-series is associated to a geographical site and to a period of time. In fact there are several attributes, some of them being constant in time, such as space or IPR, and others not, such as radiation values. We choose the description of a time-series as consisting of a series of constant attributes and a time-varying sequence of other time-varying attributes.

Gschwind Benoît, Lionel Ménard, Thierry Ranchin, Lucien Wald, Paul Stackhouse, 2007. A proposal for a thesaurus for web services in solar radiation. In Proceedings EnviroInfo 2007, O. Hryniewicz, J. Studzinski and M. Romaniuk (Eds), Shaker Verlag, vol. 1, pp. 135-142.

The first constant attribute is the unit in which the radiation value is expressed. The second is the type of radiation component, which is an enumeration: diffuse, direct, reflected, global, DNI. The third set of constant attributes deals with the IPR. It comprises the name of the provider and the relevant URL. An attribute conveys the name of the time-series given by the provider and another one the copyright text. Other attributes will convey information of dissemination issues. For example, data may be used freely or are of restricted access. We have chosen to attach these IPR attributes to the time-series and not only to the web services because of the composition of services that is foreseen. During composition, data flow are exchanged between services under the control of the collaborative information system and we believe that the controlling task is easier if IPR attributes are in the data flow. For the time being, we have not defined all these attributes; this should be done in close cooperation with service providers.

The next set of constant attributes deals with geographical location. Following BOM (2004), this set comprises the geographical coordinates and the time period of measurement which is called summarization in BOM. The associated enumeration in summarization differs slightly from WMO (2004) and BOM: 1 min, 5 min, 15 min, 1 h, 1 day, 1 week, 1 month, 1 year.

Finally, three constant attributes describe the spectral distribution. They are the two begin and end wavelengths depicting the wavelength interval over which the radiation is integrated and the unit in which these wavelengths are expressed.

It may be possible to define two other constant attributes: a scale and an offset that apply to the value as expressed in ISO 19103. We do not feel the need for these two attributes presently.

The sequence is made of a series of four time-varying attributes. One is the time corresponding to the measurement, the second is the radiation value, the third is the accuracy and the fourth one the reliability. The two latter attributes are not present in the thesaurus as there is no consensus reached up to now on that matter. This is one of the objectives of the projects IA SHC #36 and MESoR; it should be attained by the end of 2007. Though loosely defined, these attributes are present in the proposed XML schema.

Description of time is well covered by the ISO 19103, 19115 and 19118. There is no necessity for extension. Description of a date (year YYYY, month MM and day DD) and time (hour hh, minute mm and second ss) is in the form YYYY-MM-DDThh:mm:ss. Obviously, it would be better to have time expressed in decimal hour from a computational point of view. Nevertheless, we prefer to adopt this description rather than defining an extension though this would oblige most of the service providers to perform a conversion from sexagesimal system into decimal one and reciprocally.

The various systems for time may be accounted for in the ISO description. If one writes a time as hh:mm:ss±hh:mm (e.g., 18:30:43+01:00), the latter value hh:mm denotes the offset between the time used and the UT time. The offset is calculated as “local time minus UT”. In the example above, 18:30:43+01:00 is the same time than 17:30:43Z where Z (or z) is the symbol used to depict UT time. As discussed before, measurements are made sometimes in TST. Using the offset permits to provide the means to represent these data in UT time. Note that the offset between TST and UT varies every day. Thus it should be computed every day. In addition, this offset should be rounded to the minute in order to conform to ISO. This may affect measurements that are made every minute or more frequently. These cases are not the most frequent presently and we believe that we can use this representation for the time being. If necessary, an extension may be performed that may include the use of decimal hours.

A limitation of this representation is encountered for data that are aggregated for, e.g., a given month over several years. For example, customers may request mean values of irradiance for each of the 12 months of a year but integrated over 10 years or more. The ISO description or that from W3C (2004) with the extension defining part of date, e.g. a full month within a year or a full year, does not cope with this case. An extension will be necessary.

One may note that we use only one time to describe the time range in which the measurement was made. An instant combined with a duration given by the constant attribute on summarization is sufficient. However, this assumes that all providers follow the same rule in meteorological measurements: the time

Gschwind Benoît, Lionel Ménard, Thierry Ranchin, Lucien Wald, Paul Stackhouse, 2007. A proposal for a thesaurus for web services in solar radiation. In Proceedings EnviroInfo 2007, O. Hryniewicz, J. Studzinski and M. Romaniuk (Eds), Shaker Verlag, vol. 1, pp. 135-142.

given is that of the end of the measuring period. Another hypothesis is that the duration of the measurement is the same than the sampling time, which is usually the case.

We thought about another possibility to describe the instant in the time-series. We can describe the instant at which the sequence begins as a constant attribute and exploit the summarization attribute and the rank of the radiation value in the sequence to compute the measuring instant. This solution reduces the size of the message to exchange. One of the disadvantages is that it requests to have data for each instant, even if empty flag. The method for composition of services in order to detect gaps in a time-series and to fill in these gaps by invoking another web service is more difficult than if instants are provided in the sequence, even at irregular spacing. A third disadvantage is that it is more difficult to describe instant in true solar time. As already said, the difference in time between UT, or MST, and TST depends on the day; it is not constant. The only solution would be to express time only in true solar time. Experience in the SoDa Service demonstrates clearly that though physically sound and recommended by WMO in measuring solar radiation, it was not a sustainable solution as it is conflicting with the current practices.

4. XML schema

There are several possible ways for encoding metadata. The choice determines how efficiently the collaborative system may exploit and compose web services. Tendancy in composition is to define very exhaustive and precise XML schema to increase the capabilities and accuracy in composition. Reversely, a too verbose schema will discourage the services providers of using it. A trade-off should be found.

Defining an encoding is not an easy task. Obviously, it comprises an arbitrary part that depends on the choices made in implementation and exploitation. Besides the XML schema of W3C (2004) dealing with time, several XML schemas exist that are relating to meteorology and to the attributes listed above (BOM 2004; GML 2005; NOAA 2007). To our surprise, only a little of these can be exploited in our application. We believe that this is due to the fact that we are very orientated to web services and their composition and that we are dealing presently only with time-series. Description of input and output messages is of paramount importance on the efficiency of methods for adaptative composition of web services (Ponnekanti/Fox 2002; Medjahed et al. 2003). Besides the input and output messages, the web service description language (WSDL) (W3C 2001) does not comprise efficient means to describe the service in a more semantic way, i.e., what does this service. One way to overcome this limitation is to increase the semantics of the input and output messages by defining types that convey part of the semantics of the service. For example, if we define a type *irradiance*, the message does not tell much whether the data are global or direct irradiance. Defining two types *globalIrradiance* and *directIrradiance* provides a means to distinguish between two services by their messages. This is the track we are following.

Three types of attributes should be represented as types in the schema:

- the radiation parameter: irradiance, irradiation;
- the radiation component: global, direct, DNI, diffuse, reflected;
- the summarization: 1min, 5min, 15min, hourly, daily, weekly, monthly, yearly.

Our approach is to define a generic type for a time-series of radiation data *sequenceOfRadiationValues*, depicted in Fig. 1. Then this type is extended to describe the three levels (Fig. 2). An example is given by *sequenceOfHourlyGlobalIrradiance* that inherits from *sequenceOfGlobalIrradiance* that inherits from *sequenceOfIrradiance* that inherits itself from the generic type. Rather than this 3-levels cascade, it would have been more efficient to define a generic type for each of the three types of attributes and to create a new type describing a specific time-series by inheritance of three types. However, as discussed in the next section, the XML schema will be used in a certain environment in which Java is the preferred language. Java does not accept simultaneous inheritance and this solution cannot be retained.

Once the XML schema established, one possible approach to its exploitation for web services is as follows. We choose Java because we believe that it is the most appropriate tool for our development.

Gschwind Benoît, Lionel Ménard, Thierry Ranchin, Lucien Wald, Paul Stackhouse, 2007. A proposal for a thesaurus for web services in solar radiation. In Proceedings EnviroInfo 2007, O. Hryniewicz, J. Studzinski and M. Romaniuk (Eds), Shaker Verlag, vol. 1, pp. 135-142.

Firstly, the XML schema is written down with Eclipse as illustrated in the previous section. For a given service, one should define the inputs and outputs as well as the operations using the types present in the XML schema. From that, we create a WSDL file that imports the XML schema. Then, we generate the skeleton of the service in Java. Functions are implemented to complete the service. Then, it is deployed and finally, the WSDL is published for the use of the service by the collaborative information system.

5. Discussion

From 2000 to 2002, the partners in the SoDa consortium spent approximately 2.5 Meuro to build a working prototype of the SoDa Service. Then our team spent approximately 150 Keuro to make a truly operational SoDa Service with more than 20 000 users, a dozen of services providers and approximately 60 services, 10 among them being for pay. As a whole, the effort was very important (Gschwind et al. 2006). The SoDa Intelligent System (SoDa-IS) was developed to handle messages from and to web services and to invoke them (Wald et al. 2002, 2004). For that purpose, a SoDa XML schema was developed, comprising a very limited number of metadata. Each service is described by a descriptor using this schema. For example, it contains the temporal and geographical coverage of the service. The SoDa-IS has also the capability of dynamically building the GUI for each service by exploiting the service descriptor. The current applications for web services invocation, like Eclipse, do not permit to construct easily such a GUI-builder and we should spend efforts to create that application that may reach the current state of the SoDa-IS. In addition, the SoDa-IS permits to compose services in an easy way, where plans are manually built a priori in the same manner than BPEL (Van der Aalst et al. 2005). Finally, the SoDa-IS understands a set of metadata that are devoted to the presentation of the results, e.g., tables or graphs. These metadata are not part of the service descriptor; they are in the output flow from the service. That means that though all SoDa results exhibit the same SoDa frame, some arrangements are possible at the will of each provider. This possibility is very convenient but is not included in the web services description by W3C.

This discussion shows that metadata and their encoding are a major element in the SoDa-IS. Changing them will result in a in-depth re-engineering of the SoDa-IS, of message exchanges and on services provided by tiers. We are at the very beginning of this re-engineering process with many unknowns and little guidance. One of the many concerns is the relation with providers since it is the key to success for a collaborative information system. Are they ready to spend a large amount of effort to re-engineer their services? On the contrary, should we make efforts to create an interface between the legacy SoDa XML and the new one so that legacy services can be operated in the same way than before? We believe that the answer will depend upon the provider. Some of them will re-engineer their services. For the others, we may build pseudo-services, each of them encapsulating a legacy service.

The effort in re-engineering the SoDa-IS and making a new IS that is built on Open Source software, that uses known XML schemas and WSDL and is at least equivalent in functionalities to the current SoDa-IS overpasses what has been already done for the SoDa-IS in the past years. Nevertheless, it is worth making it. As the SoDa Service brings answers that are appreciated by a large number of professionals, it must evolve to survive and continue to serve efficiently the community.

Bibliography

AMS (2007) Glossary of meteorology, American Meteorology Society. Available at amsglossary.allenpress.com/glossary.

Gschwind Benoît, Lionel Ménard, Thierry Ranchin, Lucien Wald, Paul Stackhouse, 2007. A proposal for a thesaurus for web services in solar radiation. In Proceedings EnviroInfo 2007, O. Hryniewicz, J. Studzinski and M. Romaniuk (Eds), Shaker Verlag, vol. 1, pp. 135-142.

BOM (2004) XML schema for WxML markup language (weather XML), Australian Bureau of Meteorology. Available at www.bom.gov.au/bmrc/wefor/projects/b08fdp/WxML/definitions.xsd and location.xsd

CIE (1970). International Lighting Vocabulary, CIE-no 45-05-160, 1970.

Cros, S., Mayer, D., and Wald, L. (2004) *The Availability of Irradiation Data*. Report IEA-PVPS T2-04: 2004, International Energy Agency, Vienna, Austria, 29 p.

Eclipse, an open development platform. The Eclipse Foundation. Available at www.eclipse.org, with plug-ins from Lomboz distribution, available at lomboz.objectweb.org

EMP (2007). Summary of the workshop "Strategic Data for Financing Solar and Wind Power Projects", held at Ecole des Mines de Paris, Paris, France, March 16, 2007. Available at eod4invest.cma.fr/documents_en.php.

Envisolar (2004). Envisolar Project "Environmental Information for Solar Energy Industries". European Space Agency project # AO/1-4364/03/I-IW. Deliverable D1 "Integrated Service Chain Specification". Deutsches Zentrum für Luft- und Raumfahrt (DLR), Wessling, Germany.

GML (2005) Geography Markup Language, Open Geospatial Consortium. Available at schemas.opengis.net/gml/3.1.1/base.

Gschwind, B., Ménard, L., Albuisson, M., and Wald, L. (2006) Converting a successful research project into a sustainable service: the case of the SoDa Web service. *Environmental Modelling and Software*, 21, 1555-1561.

FAO (2007) Thesaurus, Aquatic Sciences and Fisheries Abstracts. Food and Agriculture Organization of the United Nations. Available at www4.fao.org/asfa/asfa.htm

INSPIRE Infrastructure for Spatial Information in the European Community. Web site: inspire.jrc.it/

IEA (2006). International Energy Agency, Solar Heating and Cooling Programme Implementing Agreement, Task 36 - Solar Resource Knowledge Management. Available at www.iea-shc.org/task36/index.html

ISO 19115 (2003): International standard. Geographic information – Metadata. First edition, ISO, Geneva, Switzerland, 140 p. Technical Corrigendum 1 (2006), 33 p.

ISO 19103 (2005): Technical Specification. Geographic information – Conceptual schema language. First edition, ISO, Geneva, Switzerland, 67 p.

ISO 19118 (2005): International standard. Geographic information – Encoding. First edition, ISO, Geneva, Switzerland, 104 p.

Medjahed, B., Bouguettaya, A., and Elmagarmid, A. K. (2003) Composing web services on the semantic web. *VLDB Journa*l, vol. 12, pp. 333-351.

NASA (2007) Surface meteorology and solar energy (SSE). Available at eosweb.larc.nasa.gov/sse/

NOAA (2007) XML schema for MADIS hydrological surface data. Available at madis-fsl.org/hydro_sfc_schema.html.

Ponnekanti S. R., Fox A. (2002) SWORD: A developer toolkit for web service composition. In Proceedings of The Eleventh World Wide Web Conference (Web Engineering Track), Honolulu, Hawaii, USA, May 7-11, pp. 83-107, 2002.

RETScreen (2007) RETScreen International. Available at www.retscreen.net/

SoDa (2007) The SoDa service for knowledge in solar radiation. Available at www.soda-is.com. European Commission-funded project IST-1999-12245 "SoDa. Integration and exploitation of networked Solar radiation Databases for environment monitoring".

UNESCO (2007) Thesaurus. Available at www2.ulcc.ac.uk/unesco.

Van der Aalst, W. M. M., Dumas, M., Ter Hofstede, A. H. M., Russel, N., Verbeek, H. M. W., and Wohed, P. (2005) Life after BPEL? *Lecture Notes In Computer Science*, 3670:35-50.

Wald, L. (2007) Solar Radiation Energy (Fundamentals). In UNESCO EOLSS (Encyclopedia of Life Support System). On-line encyclopedia available at www.eolss.net.

Gschwind Benoît, Lionel Ménard, Thierry Ranchin, Lucien Wald, Paul Stackhouse, 2007. A proposal for a thesaurus for web services in solar radiation. In Proceedings EnviroInfo 2007, O. Hryniewicz, J. Studzinski and M. Romaniuk (Eds), Shaker Verlag, vol. 1, pp. 135-142.

Wald, L., Albuisson, M., Best, C., Delamare, C., Dumortier, D., Gaboardi, E., Hammer, A., Heinemann, D., Kift, R., Kunz, S., Lefèvre, M., Leroy, S., Martinoli, M., Ménard, L., Page, J., Prager, T., Ratto, C., Reise, C., Remund, J., Rimoczi-Paal, A., Van der Goot, E., Vanroy, F., and Webb, A. (2002) SoDa: a project for the integration and exploitation of networked solar radiation databases. In: Environmental Communication in the Information Society, W. Pillmann, K. Tochtermann Eds, Part 2, pp. 713-720. Published by the International Society for Environmental Protection, Vienna, Austria.

WMO (1966). International meteorology vocabulary. WMO no 182, TP 91, World Meteorological Organization, Geneva, Switzerland.

WMO Core Profile (2004). Proposal for an XML representation of the version 0.2 of the draft WMO core profile of the ISO metadata standard, September 2004.

W3C (2001). Web Services Description Language (WSDL). Available at www.w3.org/TR/wsdl.

W3C (2004). XML Schema Part 2: Datatypes. Available at www.w3.org/TR/xmlschema-2/

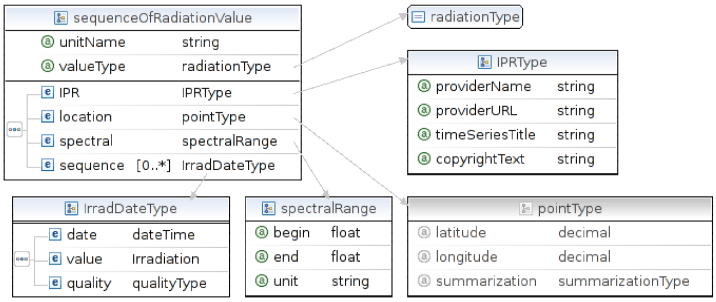

Figure 1. Schema of the generic type *sequenceOfRadiationValues*. Relations between attributes and types are shown.

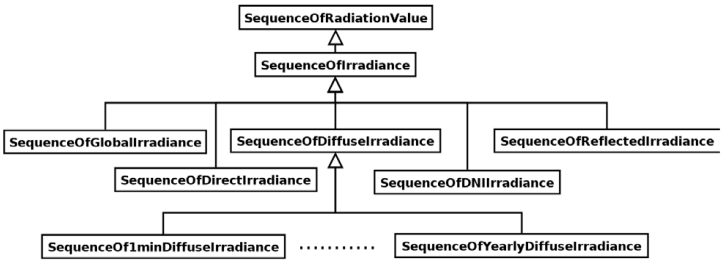

Figure 2. Diagram of inheritance of types describing the time-series of irradiance

Bibliographie

[1] Faisal Abouzaid and John Mullins. A calculus for generation, verification and refinement of bpel specifications. Electronic Notes in Theoretical Computer Science, 200 :43 – 65, 2008.

[2] S. Abramsky. Proof as processes. In Theorical Computer Science, volume 135, pages 5–9, July 1994.

[3] Sudhir Agarwal, Siegfried Handschuh, and Steffen Staab. Annotation, composition and invocation of semantic web services. Web Semantics : Science, Services and Agents on the World Wide Web, 2 :31 – 48, 2004.

[4] Vikas Agarwal, Koustuv Dasgupta, Neeran Karnik, Arun Kumar, Ashish Kundu, Sumit Mittal, and Biplav Srivastava. A service creation environment based on end to end composition of web services. In International World Wide Web Conference Committee. ACM 1-59593-046-9/05/0005, 2005.

[5] I. Budak Arpinar, Boanerges Aleman-Meza, Ruoyan Zhang, , and Angela Maduko. Ontology-driven web services composition platform. In Proceedings of the IEEE International Conference on E-Commerce Technology, 2004.

[6] Tom Barclay, Jim Gray, Steve Ekblad, Eric Strand, and Jeffrey Richter. Designing and building terraservice. IEEE Internet Computing, 10(5) :16–25, 2006.

[7] A. Barros, M. Dumas, and A. H. M. ter Hofstede. Service interaction patterns. Lecture Notes in Computer Science, 3649 :302–318, 2005.

[8] G. Bellin and P.J. Scott. On the pi-calculus and linear logic. In Theorical Computer Science, Proceedings MFPSS, volume 135, pages 11–65, July 1994.

[9] Boualem Benatallah, Marlon Dumas, Quan Z. Sheng, and Anne H.H. Ngu. Declarative composition and peer-to-peer provisioning of dynamic web services. In Proceedings of the 18th International Conference on Data Engineering (ICDE'02), 2002.

[10] D. Berardi, D. Calvanese, D. Giacomo, and M. Mecella. Reasoning about actions for e-service composition, 2003.

[11] Yves Bertot and Pierre Casteran. Interactive Theorem Proving and Program Development. SpringerVerlag, 2004.

[12] BPEL4WS, 2002. `http://www.ibm.com/developerworks/library/ws-bpelcol1/`.

[13] WSBPEL, BPEL, Web Services Business Process Execution Language BPEL, 2007. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`.

[14] F. Buckens. Considérations sur l'étude climatologique quantitative de l'habitation tropicale. Academie royale des sciences coloniales, 1956.

[15] Javier Cámara, Carlos Canal, Javier Cubo, and Antonio Vallecillo. Formalizing WSBPEL business processes using process algebra. Electronic Notes in Theoretical Computer Science, 154 :159 – 173, 2006.

[16] Mark Carman, Luciano Serafini, and Paolo Traverso. Web service composition as planning. xx :xx, 2003.

[17] Fabio Casati, Ski Ilnicki, LiJie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. Adaptive and dynamic service composition in *e*flow. In Proceedings of 12th International Conference on Advanced Information Systems Engineering, Stockholm, Sweden, March 2000. Springer-Verlag.

[18] Fabio Casati, Mehmet Sayal, and Ming-Chien Shan. Developing e-services for composing e-services. In Proceedings of the 13th International Conference on Advanced Information Systems Engineering, pages 171–186, London, UK, 2001. Springer-Verlag.

[19] Fabio Casati and Ming-Chien Shan. Dynamic and adaptive composition of e-services. Information Systems, 26 :143 – 163, 2001.

[20] E. Cerami. Web Service Essential. Distributed Applications with XML-RPC SOAP UDDI and WSDL. O'Reilly & Associates Inc., Sebastopol CA USA, 2002.

[21] Girish Chafle, Sunil Chandra, Vijay Mann, and Mangala Gowri Nanda. Orchestrating composite web services under data flow constraints. In Proceedings of the IEEE International Conference on Web Services (ICWS'05), 2005.

[22] Nizamuddin Channa, Shanping Li, Abdul Wasim Shaikh, and Xiangjun Fu. Constraint satisfaction in dynamic web service composition. In Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA'05). IEEE Computer Society, 2005.

[23] Yasmine Charif and Nicolas Sabouret. An overview of semantic web services composition approaches. Electronic Notes in Theoretical Computer Science, 146 :33 – 41, 2006.

[24] S. Cros, D. Mayer, and L. Wald. The Availability of Irradiation Data. International Energy Agency, 2004. Report IEA-PVPS T2-04 : 2004.

[25] DAML-S, 2001. `http://www.daml.org/services/`.

[26] CSIR, concil of scientific and indusrial research, 2007. `http://www.csir.co.za`.

[27] Satel-light, 2007. `http://www.satel-light.com`.

[28] Seema Degwekar, Stanley Y. W. Su, and Herman Lam. Constraint specification and processing in web services publication and discovery. In Proceedings of the IEEE International Conference on Web Services (ICWS'04), 2004.

[29] Shen Derong, Yu Ge, Yin Nan, and Nie Tiezheng. Heterogeneity resolution based on ontology in web services composition. E-Commerce Technology for Dynamic E-Business, xx :274–277, sept 2004.

[30] Van Deventer. Climatic and other disign data for evaluating heating and cooling requirements of buildings. South Affrcan Concil for Science and Industrial Research, Pretoria, South Affrica, 1971. Revision 5.0.

[31] R. DOGNIAUX. Distribution Spectrale du Rayonnement Solaire à Uccle. Institut Royale Météorologique de Belgique, 1981.

[32] E.D. Dunlop, L. Wald, and M. Suri. Solar Energy Resource Management for Electricity Generation from Local Level to Global Scale. Nova Science Publishers, 2006. ISBN : 1-59454-919-2.

[33] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. In International J. Web and Grid Services, 2005.

[34] Orna Grumberg Edmund M. Clarke and Doron A. Peled. Model Checking. MIT Press, 2000.

[35] D. Fensel and C. Bussler. The web service modeling framework WSMF. Electronic Commerce Research and Applications, 1 :113–137, 2002.

[36] G. Gardarin. Maîtriser les Bases de Données. Modèles et Langages. Edition Eyrolles, Paris, France, 1993.

[37] GEOSS, global earth observation system of systems, 2000. `http://www.earthobservations.org/geoss.shtml`.

[38] Malik Ghallab, Craig Knoblock Isi, Scott Penberthy, David E Smith, Ying Sun, and Daniel Weld. Pddl - the planning domain definition language. Technical report, Ecole Nationale Superieure D'ingenieur des Constructions Aeronautiques, 1998.

[39] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. Congolog, a concurrent programming language based on the situation calculus, 2000.

[40] B. Gschwind, L. Menard, M. Albuisson, and L. Wald. Three years of experience with the SoDa web service delivering solar radiation information : lessons learned and perspectives. In J. Racek Eds J. Hrebicek, editor, Proceedings of the 19th International Conference on Informatics for Environmental Protection, pages 95–102, Czech Republic, 2005. Masaryk University in Brno.

[41] Benoit Gschwind, Lionel Menard, Michel Albuisson, and Lucien Wald. Converting a successful research project into a sustainable service : The case of the SoDa web service. Environmental Modelling & Software, 21(11) :1555–1561, nov 2006.

[42] Benoit Gschwind and Lucien Wald. A proposal for a thesaurus for web services in solar radiation. 2007.

[43] Claudio Guidi, Roberto Lucchi, and Manuel Mazzara. A formal framework for web services coordination. Electronic Notes in Theoretical Computer Science, 180 :55 – 70, 2007.

[44] Seyyed Vahid Hashemian and Farhad Mavaddat. A graph-based approach to web services composition. In Proceedings of the 2005 Symposium on Applications and the Internet (SAINT'05). IEEE Computer Society, 2005.

[45] Peter Höfner and Florian Lautenbacher. Algebraic structure of web services. Electronic Notes in Theoretical Computer Science, 200 :171 – 187, 2008.

[46] Alfred Horn. On sentences which are true of direct unions of algebras. Journal of Symbolic Logic, 16 :14–21, 1951.

[47] San-Yih Hwang, Haojun Wang, Jian Tang, and Jaideep Srivastava. A probabilistic approach to modeling and estimating the qos of web-services-based workflows. Information Sciences, 177 :5484 – 5503, 2007.

[48] IEA, internationnal energy agency, 2009. http://www.iea.org/.

[49] 2009. http://www.json.org/.

[50] R. Khalaf, A. Keller, and F. Leymann. Business processes for web services : Principles and applications. IBM Systems Journal, 45 :425–446, 2006.

[51] Jong Myoung Ko, Chang Ouk Kim, and Ick-Hyun Kwon. Quality-of-service oriented web service composition algorithm and planning architecture. The Journal of Systems and Software, xxx :xxx – xxx, 2008.

[52] I. Kotsiopoulos, J. Keane, M. Turner, P. J. Layzell, and F. Zhu. Ibhis : integration broker for heterogeneous information sources. In Proceedings of the 27th Annual International Computer Software and Applications Conference, pages 378–384, Washington, DC, USA, 2004. IEEE Computer Society.

[53] Ugur Kuter, Evren Sirin, Bijan Parsia, Dana Nau, and James Hendler. Information gathering during planning for web service composition. Web Semantics : Science, Services and Agents on the World Wide Web, 3 :183 – 205, 2005.

[54] Sven Lämmermann. Runtime service composition via logic-based program synthesis. PhD thesis, Royal Institute of Technology, June 2002. Department of Microelectronics and information Technology, Royal Institute of Technology, Stockholm, Sweden.

[55] Rubén Lara, Dumitru Roman, Axel Polleres, and Dieter Fensel. Web Services, chapter A Conceptual Comparison of WSMO and OWL-S, pages 254 – 269. Springer Berlin, 2004.

[56] Rob Lemmens, Andreas Wytzisk, Rolf de By, Carlos Granell, Michael Gould, and Peter van Oosterom. Integrating semantic and syntactic descriptions to chain geographic services. IEEE Internet Computing, 10(5) :42–52, 2006.

[57] V. R. Lesser. Cooperative multiagent systems : A personal view of the state of the art. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, 11 :133–142, 1999.

[58] Patrick Lincoln. Deciding provability of linear logic formulas. In Proceedings of the workshop on Advances in linear logic, pages 109–122, New York, NY, USA, 1995. Cambridge University Press.

[59] Roberto Lucchi and Manuel Mazzara. A pi-calculus based semantics for ws-bpel. The Journal of Logic and Algebraic Programming, 70 :96 – 118, 2007.

[60] Yue Ma and Chengwen Zhang. Quick convergence of genetic algorithm for qos-driven web service selection. Computer Networks, 52 :1093 – 1104, 2008.

[61] T. Madhusudan and N. Uttamsinghn. A declarative approach to composing web services in dynamic environments. Decision Support Systems, 41 :325–257, 2006.

[62] David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia Sycara. Bringing semantics to web services : the owl-s approach. In Semantic Web Services and Web Process Composition, volume 3387, pages 26–42. Springer Berlin / Heidelberg, 2005.

[63] Drew McDermott. Estimated-regression planning for interactions with web services. In Proceedings of the 6th International Conference on IA Planning and Scheduling, Toulouse, France, 2002. AAAI Press.

[64] Sheila A. McIlraith and Tran Cao Son. Adapting golog for composition of semantic web services. In Proceedings 8th International Conference on Knowledge Representation and Reasonning, Toulouse, France, April 2002.

[65] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. IEEE Intelligent Systems, 3(16) :46–53, 2001.

[66] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid. Composing web services on the semantic web. The VLDB Journal, 12 :333–351, 2003.

[67] MESOR, management and exploitation of solar resource knowledge, 2007. `http://www.mesor.org`.

[68] Global meteorological database for engineers, planners and education. Meteonorm version 5.0, 2003. `http://www.meteonorm.com`.

[69] S. Narayanan and S. A. McIlraith. Simulation, verification and automated composition of web services. In Proceedings of the 11th International Conference on World Wide Web, pages 77–88, Honolulu, Hawaii, USA, 2002. ACM Press, New York, NY, USA.

[70] OGC, open geospatial consortium, 1994. `http://www.opengeospatial.org`.

[71] OMM organisation météorologique mondiale, 2000. `http://www.wmo.ch/pages/index_fr.html`.

[72] World Meteorological Organization, editor. Meteorological Aspects Of The Utilization of Solar Radiation as an Energy Source. World Meteorological Organization, 1981.

[73] OWL Ontology Web Language, 2004. `http://www.w3.org/TR/owl-ref/`, `http://www.w3.org/TR/owl-features/`.

[74] OWL-S Ontology Web Language - Semantics, 2005. `http://www.daml.org/services/owl-s/1.0/owl-s.html`.

[75] Claus Pahl and Yaoling Zhu. A semantical framework for the orchestration and choreography of web services. Electronic Notes in Theoretical Computer Science, 151 :3 – 18, 2006.

[76] W. Palz. ESRA, European Solar Radiation Atlas, volume I et II. DG Science, second and extended edition, 1984.

[77] C. Petlz. Web services orchestration and choreography. Computer, 36 :46 – 52, 2003.

[78] G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[79] Shankar R. Ponnekanti and Armando Fox. SWORD : A developer toolkit for web service composition. In Proceedings of the 11th International World Wide Web Conference, Honolulu, HI, USA, 2002.

[80] Jinghai Rao, Peep Küngas, and Mihhail Matskin. Application of linear logic to web service composition. In Proceedings of 1st International Conference on Web Services, Las Vegas, USA, june 2003.

[81] Jinghai Rao, Peep Küngas, and Mihhail Matskin. Logic-based web services composition : from service description to process model. In Proceedings of the 2004 International Conference on Web Services, San Diego, USA, July 2004. IEEE.

[82] Jinghai Rao, Peep Küngasa, and Mihhail Matskinb. Composition of semantic web services using linear logic theorem proving. Information Systems, 31 :340 – 360, 2006.

[83] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, San Diego, California, USA, July 2004.

[84] Jan Remund, Stefan Kunz, and Ralf Lang. METEONORM, Global Meteorological Database for solar energy and applied climatology. Meteotest, 1999.

[85] Jan Remund, Esther Salvisberg, and Stefan Kunz. METEONORM, Energie solaire et météorologie Notions de based. Meteotest, 1995.

[86] Philippe Roussel. Prolog, manuel de référence et d'utilisation, 1975. Groupe d'Intelligence Artificielle, UER Marseille-Luminy.

[87] Stuart Russell and Petter Norvig. Artificial Intelligence A Modern Approach. Pearson Education, Inc., 2003. Second Edition.

[88] John T. Sample, Roy Ladner, Lev Shulman, Elias Ioup, Fred Petry, Elizabeth Warner, Kevin Shaw, and Frank P. McCreedy. Enhancing the US Navy's GIDB portal with web services. IEEE Internet Computing, 10(5) :53–60, 2006.

[89] Hans Schuster, Dimitrios Georgakopoulos, Andrzej Cichocki, and Donald Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In Proceedings of 12th International Conference on Advanced Information System Engineering, Stockholm, Sweden, June 2000. Springer-Verlag.

[90] Mithun Sheshagiri, Marie desJardins, and Timothy Finin. A planner for composing services described in daml-s. In Proceedings of The 13th International Conference on Automated Planning & Scheduling, 2003.

[91] Yuliang Shi, Liang Zhang, Bing Liu, Fangfang Liu, Lili Lin, and Baile Shi. A formal specification for web services composition and verification. In Proceedings of the 2005 The Fifth International Conference on Computer and Information Technology (CIT'05). IEEE Computer Society, 2005.

[92] Evren Sirin, James Hendler, and Bijan Parsia. Semi-automatic composition of web services using semantic descriptions. In Proceedings of Web Services : Modeling, Architecture and Infrastructure Workshop in conjonction with ICEI2003, 2002.

[93] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau. HTN planning for web service composition using SHOP2. Web Semantics : Science, Services and Agents on the World Wide Web, 1 :377 – 396, 2004.

[94] S.J.Woodman, D.J.Palmer, and S.K.Shrivastava. Notations for the specification and verification of composite web services. In Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conf (EDOC 2004), 2004.

[95] SoDa, solar data, 2000. `http://www.soda-is.com`.

[96] Monika Solanki, Antonio Cau, and Hussein Zedan. Augmenting semantic web service descriptions with compositional specification. In Proceedings of the 13th international conference on World Wide Web (WWW'04), pages 544–552, New York, NY, USA, 2004. ACM.

[97] P. Stackhouse, D. Renne, R. Perez, R. Meyer, L. Wald, and M. Suri. Towards designing an integrated earth observation system for the provision of solar energy resource and assessment. In Proceedings of IEEE IGARSS Symposium 2006, Washington, DC, USA, 2006. IEEE. Denver, Colorado, USA, 31 July 4 August 2006.

[98] Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. Web Semantics : Science, Services and Agents, 1 :27 – 46, 2003.

[99] Stefan Tai, Rania Khalaf, and Thomas Mikalsen. Composition of coordinated web services. In H.-A. Jacobsen (Ed.) : Middleware 2004, editor, IFIP International Federation for Information Processing 2004, pages 294 – 310, 2004.

[100] H. Cormen Thomas, Charles E. Leiserson, and Ronald L. Rivest. Introduction to Algorithms. MIT Press, 1996.

[101] UDDI Universal Description, Discovery and Integration, 2005. `http://www.uddi.org/`.

[102] W. M. MP. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, H. M. W. Verbeek, and P. Wohed. Life after BPEL ? Lecture Notes in Computer Science, 3670 :35–50, 2005.

[103] Mirko Viroli. Towards a formal foundation to orchestration languages. Electronic Notes in Theoretical Computer Science, 105 :51 – 71, 2004.

[104] L. Wald. Data Fusion. Definitions and Architectures. Fusion of Images of Different Spatial Resolutions. Les Presses de l'Ecole des Mines, Paris, France, 2002.

[105] Lucien Wald, Michel Albuisson, C. Best, C. Delamare, D. Dumortier, E. Gaboardi, A. Hammer, D. Heinemann, R. Kift, S. Kunz, M. Lefevre, S. Leroy, M. Martinoli, L. Menard, J. Page, T. Prager, C. Ratto, C. Reise, J. Remund, A. Rimoczi-Paal, E. Van der Goot, F. Vanroy, and A. Webb. SoDa : a project for the integration and exploitation of networked solar radiation databases. In Environmental Communication in the Information Society, pages 713–720, Vienna, Austria, 2002. the International Society for Environmental Protection. W. Pillmann, K. Tochtermann Eds.

[106] Richard J. Waldinger. Web agents cooperating deductively. In FAABS '00 : Proceedings of the First International Workshop on Formal Approaches to Agent-Based Systems-Revised Papers, pages 250–262, London, UK, 2001. Springer-Verlag.

[107] WMO, World Meteorological Organisation. `http://www.wmo.ch`.

[108] 1961 - 1990 global climate normals (clino). Published by National Climatic Data Center, 1998.

[109] WSCL Web Service Choregraphy Description Langage, November 2005. `http://www.w3.org/TR/ws-cdl-10/`.

[110] WSDL, web service description language, 2001. `http://www.w3.org/TR/wsdl`.

[111] WSMO, Web Service Modeling Ontology, 2006. `http://www.wsmo.org/TR/d2/v1.3/`.

[112] Webservicex. `http://www.webservicex.net`.

[113] Dan Wu, Evren Sirin, James Hendler, Dana Nau, and Bijan Parsia. Automatic web services composition using SHOP2. In Workshop on Planning for Web Services, Trento, Italy, June 2003.

[114] XML-RPC, XML Remote Procedure Call, 1999. `http://www.xmlrpc.com/spec`.

[115] XSD XML Schema, 2005. `http://www.w3.org/XML/Schema`.

[116] Peng Yuea, Liping Dia, Wenli Yanga, Genong Yua, and Peisheng Zhao. Semantics-based automatic composition of geospatial web service chains. Computers & Geosciences, 33 :649 – 665, 2007.

[117] F. Zhu, M. Turner, I. Kotsiopoulos, K. Bennett, M. Russell, D. Budgen, P. Brereton, J. Keane, P. Layzell, M. Rigby, and J. Xu. Dynamic data integration using web service. In Proceedings of IEEE International Conference on Web Services, pages 262–269, Washington, DC, USA, 2004. IEEE Computer Society.

[118] Roger Zimmermann, Wei-Shinn Ku, Haojun Wang, Amir Zand, and Jean-Pierre Bardet. A distributed geotechnical information management and exchange architecture. IEEE Internet Computing, 10(5) :26–33, 2006.